MICROPROCESSOR APPLICATION
IN A
FIRE CONTROL SYSTEM

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

LCDR. JAGDISH ANAND I. N.

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

AUGUST 1977

POST GRADUATE OFFICE
Submitted on 6.8.77

CERTIFICATE

Certified that this work on 'Microprocessor Application in a Fire Control System' has been carried under my supervision and that this has not been submitted elsewhere for a degree.

M.S. Krishnamoorthy

Dr. M.S. Krishnamoort
Lecturer
Computer Centre,
I.I.T. Kanpur.

August, 1977

ACKNOWLEDGEMENTS

To work with Dr. M.S. Krishnamoorthy, my thesis advisor, has been a lesson in sincerity, dedication and logical thinking. His immense patience and inspiring guidance have made working with him a pleasure. It is, therefore, with a deep sense of gratitude that I acknowledge my gratefulness to him.

To Lt. Gian Prakash, I.N., goes a note of thanks for his assistance in the preparation of the numerous figures included in this thesis. Thanks also goes to Mr. Abraham for an outstanding job in the typing of this thesis.

Finally, a note of thanks to my wife, Amita, who more than anyone else, encouraged and cheered me when this work progressed slowly.

Anand
LCDR

IIT Kanpur
August 1977

– J. Anand

TABLE OF CONTENTS

LIST OF FIGURES

SYNOPSIS

'Microprocessor Application in a Fire Control System'. A thesis submitted in Partial Fulfilment of the requirements for the degree of MASTER OF TECHNOLOGY by Lieutenant Commander Jagdish Anand I.N., to the Department of Electrical Engineering, Indian Institute of Technology, KANPUR, in August, 1977.

This thesis describes the Fire Control Problem as applicable to anti-aircraft firing from a naval vessel. Existing systems use analog computation for the solution of this problem. A digitalised Error Predictor, using an INTEL 8080 Microprocessor, to solve the basic fire control problem, is discussed. The various assumptions, approximations and corrections made are also mentioned. The mathematics of computation of time of flight is outlined and its computation is presented in details.

A greater accuracy in the system is introduced by incorporating in the predictor an error correction technique, wherein, an error function for range, elevation and bearing is continuously computed and applied to the computed future co-ordinates to give a more accurate prediction. This enhances the accuracy of the system and increases the probability of hit. The computation of the error function is made possible by storing the predicted future co-ordinates at very short intervals of time and then comparing these with the co-ordinates of the actual aircraft positions after the elapse of time of

flight intervals to get the error values. These error values are put on stack and are used to fit a curve from which the error functions are computed.

A new dimension to the problem is the implementation of fire control system using Microprocessors. An introduction to Microprocessor architecture, constraints due to their use, description of INTEL 8080 are included as part of the thesis. The Microprocessor system software is dealt with in details. Assembly language of INTEL 8080, with special reference to instructions extensively used is discussed. Special algorithms due to the use of Microprocessors for computation of square roots and curve fitting are also included. Fortran Programs and the 8080 assembly language program are included in the Appendices.

Results show that errors in prediction are considerably reduced and that a Microprocessor can easily be adapted to the solution of a real time application. A dedicated Microprocessor, supported by a small amount of specialised interface hardware and a well designed software support, is shown to offer a viable alternative to special purpose discrete logic systems. The low cost of microprocessors provide a significant economic advantage to systems of this type.

INTRODUCTION

The main aim of any weapon control system is to fire a shell(s) from the gun(s) that the system controls so as to destroy the approaching target. Underlying this situation is a certain element of probability : the target position at the time of hit must be predicted, and there is a certain interval between the launching and the hit-called the time of flight - during which the projectile is wholly under the influence of natural phenomenon. The computation of the time of flight and the accurate prediction of the target future position, in essence, constitute the fire control problem.

Existing systems of the Navy use analog computers for the solution of this problem. The aim of this thesis is to suggest a digitalised error predictor, using an INTEL 8080 Microprocessor as its main computing element to solve the fire control problem. The predictor described incorporates an error correction technique which enhances the accuracy of the system and increases the probability of hit. The Microprocessor system software is dealt with in full details and the system program in the assembly language of 8080 is also presented.

The thesis is divided into six chapters. Chapter 1 describes the Fire Control Problem. It deals with the

assumptions pertaining to the movement of the target, the motion of the projectile, choice of the co-ordinate system and outlines a method for the computation of time of flight. Chapter 2 goes into the details of the time of flight computations. It discusses the corrections that must be applied to the standard trajectory solution and outlines the mathematics of time of flight computations. The time of flight routine used in the system is discussed.

Chapter 3 discusses the improved prediction technique. It outlines the general representation of the system and lists the inputs to and outputs from the predictor. It describes the computation of the prediction error and the various techniques to correct this error. A detailed description of the error correction technique used is presented. Chapter 4 deals with the system design. It presents an introduction to general Microprocessor architecture, microprocessor imposed constraints, the choice of a microprocessor for the system, and a brief description of Intel 8080 microprocessor. It discusses the system requirements and outlines the proposed system using 8080 and other Hardware modules.

Chapter 5 is a detailed chapter on System Software. In this the 8080 assembly language with particular reference to instructions extensively used in programming the system are explained. The special algorithms used, the memory

requirement and timing constraints are discussed. A system flow chart is presented and explained.

Chapter 6 concludes the thesis and outlines the scope for further work. The Fortran Program with sample results and the 8080 Assembly Language Programme are listed in the Appendices.

1. FIRE CONTROL PROBLEM

1.1 General Introduction

Fire Control Problem involves the launching of a projectile from a weapon station (that may be moving) at a target (that may also be moving) so as to score a hit on the target. To achieve this, the first object in all cases of gunnery against a moving target is to fire a projectile in a direction and at a time to cause the projectile and the target to reach a certain point simultaneously. This point is called the future position of the target and will be denoted by F. The position of the target at the moment of firing is called the present position and will be denoted by P. The calculations necessary to achieve this object constitute the prediction problem in its widest sense, and the computing device that solves this prediction problem is called a PREDICTOR.

An important function for any prediction system is that of computing the time of flight t_f, the interval between the launching of the projectile and the hit, during which the projectile is wholly under the influence of natural phenomenon. To compute this accurately it is necessary to have complete knowledge as to how both the target and the projectile will move during this interval. Because complete

information can never be available (and if it were, the necessary computations would be so complicated as to be impracticable), it is necessary to make certain assumptions about the movement of both the target and the projectile.

1.2 Assumptions Pertaining to Movement of Target

The predicted motion of target is based on measurement of its motion prior to the moment of firing and on assumptions as to how it will move during the time of flight. The problem in this case is how to find the spherical polar co-ordinates (R_f, B_f, S_f) of the future position F, allowing for target movement during the time of flight t_f. The information available consists of the spherical polar co-ordinates (R_p, S_p, B_p) of the present position P and their corresponding rates of change ($\dot{R}$, $\dot{S}$, $\dot{B}$). These are assumed to be available continuously.

Systems that were designed till now have made use of one of the following set of assumptions as to target movement during the time of flight.

a) The target travels at a constant speed in a straight line at a constant height.

b) The target travels at constant speed in a straight line, but not necessarily at a constant height.

c) The target travels at constant speed, with constant rate of change of height and constant rate of change of course.

d) The target has constant vector acceleration.
Of these four sets, though the condition in (a) is the most restrictive it is the one that is most commonly fulfilled in normal flight. It can usually be expected to hold except when the target is taking avoiding action of some kind or manoeuvring for attack.

Assumption (b) is the most general for which the target vector velocity is constant. It is the most common assumption in use as the system is operated with a radar and hence R_p, $\dot{R}$, B_p, $\dot{B}$, S_p, $\dot{S}$ are continuously available. Assumptions (c) and (d) permit a curved target track, and although more accurate, involve measurement of second derivatives of suitable co-ordinates which are not accurately measurable in the present existing systems in the country.

For the design considered, assumption (b), i.e. the target is flying in a straight line at constant speed, but not necessarily at a constant height, is assumed.

The success of the prediction will depend very much on the fulfilment of this assumption by the target. A pilot who goes into or straightens out of a turn or dive during the time of flight, whether this is done by chance or is deliberate avoiding action, can always reduce very considerably the chance of a hit, however elaborate the prediction formulae.

1.3 Motion of the Projectile

The predicted movement of the projectile is based partly on experiment and partly on theory, and the accuracy of this prediction is naturally dependent on the reliability of both of these. It is even more dependent however on the reproduction in practice of the conditions under which the experiments are conducted or on which the theory is based. These conditions are liable to variation in many details, such as the state of the atmosphere, the speed and direction of the wind at various heights, wear of the gun at the instant of firing, abnormality of the projectile etc. It is possible to allow for some of these, but their more or less infinite variety renders complete correction impracticable.

The most important effect for which allowance must be made in prediction is the target movement. Thus, this problem has been considered separately and corrections due to ballistic abnormalities are added on to it subsequently. The computation of the time of flight therefore incorporates standard conditions in vacuo with standard muzzle velocity of the gun. The corrections applied are discussed under a separate heading.

1.4 Choice of Co-ordinate System

The three basic sets of co-ordinates are :

(a) Spherical polar co-ordinates - slant range (R), bearing (B) and angle of sight (S)

(b) Cylindrical polar co-ordinates: plan range (r), bearing (B) and height (H)

(c) Cartesian co-ordinates: eastings (E), northings (N) and height (H).

Referring to Figure 1.1, it is seen that they are connected by the formulae,

$$r = R \text{ Cos } S$$

$$H = R \text{ Sin } S$$

$$E = r \text{ Sin } B = R \text{ Cos } S \text{ Sin } B$$

$$N = r \text{ Cos } B = R \text{ Cos } S \text{ Cos } B$$

Fire control systems are distinguished by the particular set of co-ordinates that are used for the main prediction computations. The system designed uses the spherical polar co-ordinates as spherical polars are the quantities that are measured by a conventional director system. Further the two angles required for aiming a gun correctly (B_f and ϕ) are in effect spherical polar co-ordinates, so if these have not been used for prediction it will be necessary to convert back.

1.5 Computation of Time of Flight

In order to predict the future position the time of flight must be known, but the time of flight cannot be calculated till the future position has been found. The problem is diagramatically represented in Figure 1.2 and can be represented as a set of two equations,

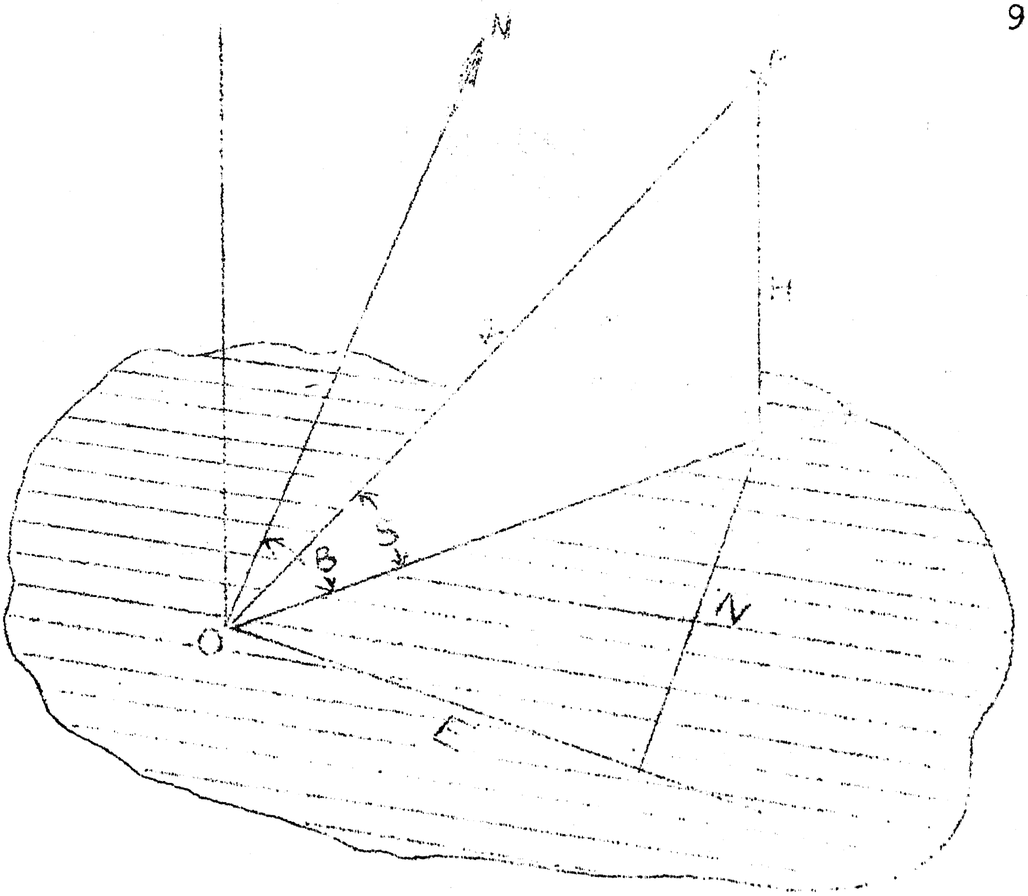


Figure 1.1
Co-ordinate Systems Representation

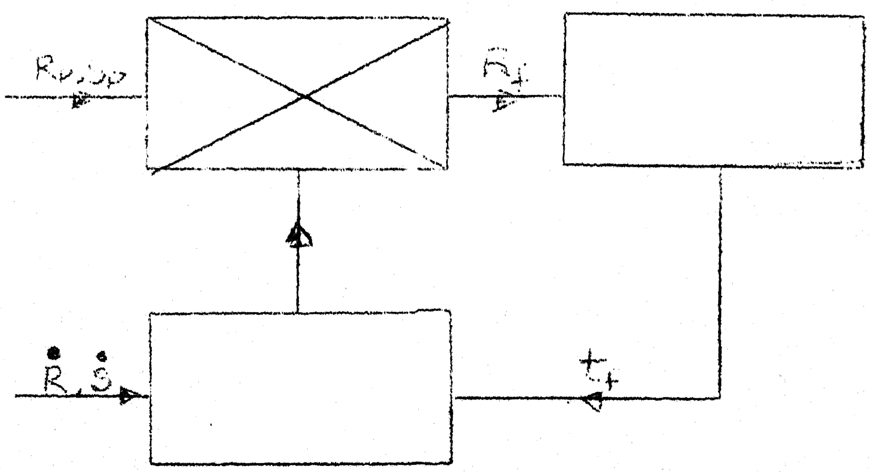


Figure 1.2
Cycleof Operation in Computation of Time of Flight

$$F = f_1 \ (P, \ t_f) \qquad (1)$$

$$t_f = f_2 \ (F) \qquad (2)$$

The first of these is a kinematical equation determined by the target. It gives the position of the target after a time t_f and is of course in very much simplified form here. It should in fact be two equations, to give R_f and S_f and it involves not only the co-ordinates of P but also their derivatives. Equation (2) is based on the motion on the motion of the shell and is therefore ballistic. These two equations can not be explicitly solved and thus an iterative method is used to solve these. A trial value of $t_f = t_p$ (the time of flight to the present position P, approximated to present range/muzzle velocity), is selected. This value of t_f is used to obtain a future position F by means of equation (1). The co-ordinates (R_f, S_f) of this position are substituted in equation (2) to obtain a new trial value t_f. This cycle of operation is repeated until no appreciable changes are obtained in the values of R_f and t_f, which are then the required values. It can be proved that this method is always convergent, and for the cases studied, the algorithm used has been such that the final values are obtained in ≤ 10 iterations at start and in ≤ 7 iterations for each subsequent values at intervals of 0.1 or 0.2 second.

2. DETAILED TIME OF FLIGHT COMPUTATION

2.1. Mathematics of Computation of Time of Flight and Tangent Elevation

In the Figure 2.1, GT is the initial direction of motion of shell and TF is a vertical from T (drop height or gravity drop).

Thus the future position F can be defined by means of the lengths GT and TF. For a general trajectory these are determined from two differential equations. Owing to the complicated form of the air resistance law these equations can not be solved explicitly. If however the air resistance is neglected or is taken to be proportional to the velocity, explicit solutions can be obtained and although these are not accurate enough to be used as approximations in the general trajectory, they do provide suggestions as to the forms that emperical formulae might take [1]. For the system under consideration, for simplicity, the trajectory is considered in vacuo, with no air resistance. Under these conditions $GT = Vt$, and $TF = \frac{1}{2} gt^2$, where V is the initial velocity (muzzle velocity).

From Triangle GFT, we have

$$\text{Sin } E = \frac{TF}{GT} \text{ Cos } S_f = \frac{gt}{2V} \text{ Cos } S_f$$

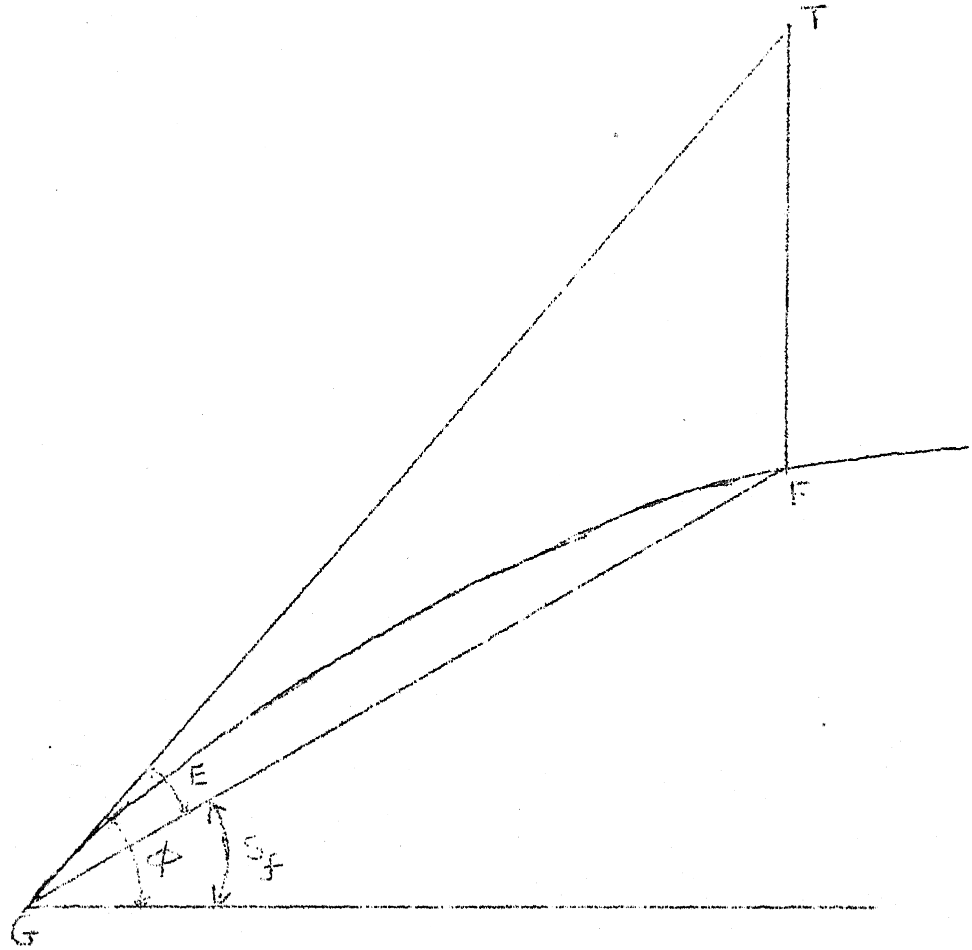


Figure 2.1
General Trajectory Representation

As the tangent elevation E is small, it is equal to $f(t_f)$ Cos S_f. The function $f(t_f)$ is of course the tangent elevation E_o for zero angle of sight and time of flight t_f. Thus,

$$\text{SIN } E = E = E_o \text{ Cos } S_f$$

In the case where the air resistance is proportional to velocity, say kV per unit mass where V is the instantaneous velocity and k is a constant, it can be proved that

$$GT = \frac{V}{K}\,(1 - e^{-kt})$$

$$TF = \frac{g}{k^2}\,(kt - 1 + e^{-kt})$$

and similarly applying the sine formula, for small values of E we have

$$E = E_o \text{ Cos } S_f$$

Although the justification of this approximation is valid only in the case of vacuo or when air resistance is proportional to velocity, it is found that a similar formula can be used with a general resistance law [1]. A graph showing the tangent elevation for different values of S_f of an existing gun is shown in Figure 2.2. The dotted curve shows values of E_o Cos S_f. It can thus be seen that this approximation is reasonably justified as the variations from actual is nominal.

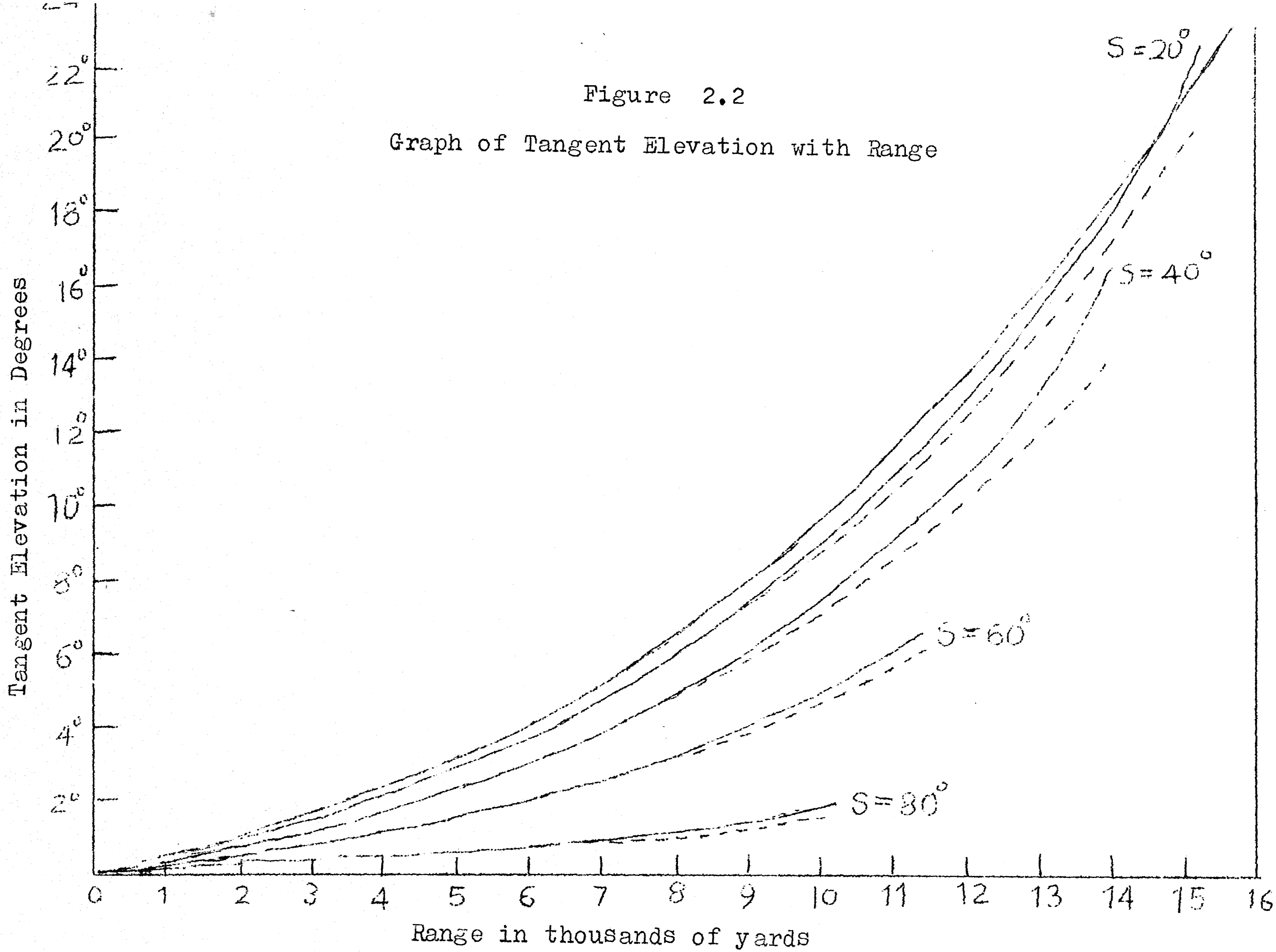


Figure 2.2

Graph of Tangent Elevation with Range

From Figure 2.1, we have,

$$R_f = GF = GT \frac{\cos \phi}{\cos S_f} \quad (\phi = S_f + E)$$

Now $\frac{\cos \phi}{\cos S_f} = \frac{\cos (S_f + E)}{\cos S_f} = \cos E - \tan S_f \sin E$

As E is small this can be approximated to be

$$\frac{\cos \phi}{\cos S_f} = 1 - \tan S_f E_o \cos S_f$$

$$= 1 - E_o \sin S_f$$

Therefore, $R_f = GT\ (1 - E_o \sin S_f)$

$$= Vt\ (1 - \frac{gt}{2V} \sin S_f)$$

i.e. $2\ R_f \doteq 2\ V_t - gt^2 \sin S_f$

Therefore, $g \sin S_f\ t^2 - 2\ Vt + 2\ R_f = 0$ (1)

This gives a quadratic in t and corresponds to the second equations in the time of flight computation as depicted in the previous chapter. Restricting to first order approximations the first equation is

$$R_f = R_p + \dot{R}\ t_f \qquad (2)$$

From equations (1) and (2) we iteratively get the time of flight and future co-ordinates of the target. The future co-ordinates for elevation and bearing are obtained by computing the vertical and azimuth deflection.

(D_v) and (D_a) respectively. These are given by the relation [1]

$$\mathrm{Sin}(D_v) = \frac{R_p}{R_f} \dot{S}\, t_f$$

$$\mathrm{Sin}(D_a) = \frac{R_p \;\mathrm{Cos}\; S_p}{R_f \;\mathrm{Cos}\; S_f} \dot{B}\, t_f$$

$$S_f = S_p + D_v$$

and

$$B_f = B_p + D_a$$

These co-ordinates are then corrected for the following

(a) Corrections for non-standard ballistic conditions

(b) Drift corrections

(c) Corrections due to the effect of wind on shell

(d) lateral and vertical convergence due to the physical displacement between the directing system and the guns (Refer Appendix A).

These corrected co-ordinates are then fed to the guns as 'Gun orders' and the gun is therefore 'trained' and 'layed' to fire at the future position of the target.

2.2 Correction for non-standard ballistic Conditions

These can be separated into internal and external effects. The former include wear of the bore and variation of charge temperature, whose effect is to change the muzzle velocity. The latter arise on account of variation of external temperature, humidity and barometric pressure and cause a change in ballistic coefficient.

Referring to Figure 2.1 and equations in last section it is seen that a change in the muzzle velocity V produces a proportional change in GT, but no change in TF. This is exactly true in case of motion in vacuo and with resistance proportional to the velocity and is approximately true for the general trajectory. The change is represented as a percentage = δV. The resistance of the air to the motion of the shell depends on the velocity relative to the air and also on the shape and calibre of the shell and the density of the air. It is in fact inversely proportional to the ballistic coefficient, defined as

$$Mf / K\sigma T d^2$$

where K = coefficient of shape

σ = coefficient of steadiness

d = calibre

M = Mass of shell

f = a factor which allows for the variation of the air density with height (unity for zero height)

T = coefficient of tenuity, the density of the air at sea level. It is a function of height. T depends on the barometric pressure, absolute temp., and humidity at sea level and can be calculated when these are known. The basic trajectory is calculated [1,2] for standard values of

pressure, temperature and humidity, and variations from these values when set in the predictor are combined to give a change δC in C (expressed as a percentage). Limiting the corrections to first order approximations, the corrections to time of flight and elevation are linear functions of δV and δC i.e.

$$\delta t = A\delta V + B\delta C$$

and

$$\delta S_f = A'\delta V + B'\delta C$$

where A, B, A', B' are functions of the co-ordinates of F. Since these corrections are small, relatively crude emperical approximations can be made. For the system under design these were simulated using random number generation procedures.

2.3 Drift Correction

This is due to the spinning action given to a shell due to the rifling in the muzzle of the gun. This causes a precession in the shell making the axis of the shell point away from the direction of motion. Drift is usually represented as an angular displacement, and is corrected by applying an equal and opposite deflection in azimuth or in lateral plane. The azimuth deflection for drift is a function of the time of flight t_f but is independant of ϕ. The function $f(t_f)$ is not in fact linear, but since drift is a small angle, a linear approximation is usually adequate. The azimuth deflection for drift is approximately proportional to tangent elevation

at zero angle of sight [1]. The corresponding deflection in the future lateral sight-plane is therefore given by

$$\text{Drift} = \text{Drift constant} \times E_o \text{ Cos } S_f = \text{Drift constant} \times \text{Tangent Elevation}$$

2.4 Corrections due to Wind

In considering the wind effects it is assumed that the wind is constant and horizontal. The direction from which it is blowing is usually represented by the angle WI (called the Wind Inclination) measured clockwise from the vertical plane of fire.

If W = Wind speed

$Wr = W \text{ Cos } (WI)$ is the component along the vertical plane of fire

$Wl = W \text{ SIN } (WI)$ is the component across the vertical plane of fire.

It is found that due to the wind the appropriate corrections are [1].

(i) $\delta R = Wr\ t_f \text{ Cos}\phi$ (to range)

(ii) $\delta\phi = -(Wr\ t_f \text{ Sin}\phi)/R_f$ (to elevation)

The range correction is affected by corrections $(\partial t/\partial R)\delta R$ and $(\partial\Psi/\partial R)^*\delta R$ to time of flight and elevation respectively. These work out to $\delta t_w = Wr\ t_f \text{ Cos } S_f/V$ and $\delta dv_w = -(Wr \text{ Cos } S_f/V \text{ Sin }\phi \text{ Cos}\phi)$.

Like the other corrections these are also relatively small, so that the various functions of F can be represented by emperical approximations, such as linear functions of range or time of flight, obtained usually by graphical methods.

In considering wind effects, we have assumed constant wind speed and direction at all points of the trajectory. Through sufficient metrological information an 'equivalent constant wind' (defined as the constant wind which will produce the same displacement in magnitude and direction as actual air motion) is determined and used in the computations. In all the cases considered wind speed considered is the equivalent constant wind.

2.5 Time of Flight Routine

A flow chart for the computation of time of flight is shown in Figure 2.3. This TIME routine is called by the main Prediction Error Program to compute the time of flight and the future point of impact co-ordinates of the shell and the aircraft, if the firing was to take place at the present instant. The time of flight computation is done iteratively within the allowable tolerance of 0.05 seconds in time and 1 foot in range. For the cases studied this routine is found to converge in $\leq$ 10 iterations. A safety margin is provided by allowing the routine to run to a maximum of 15 iterations.

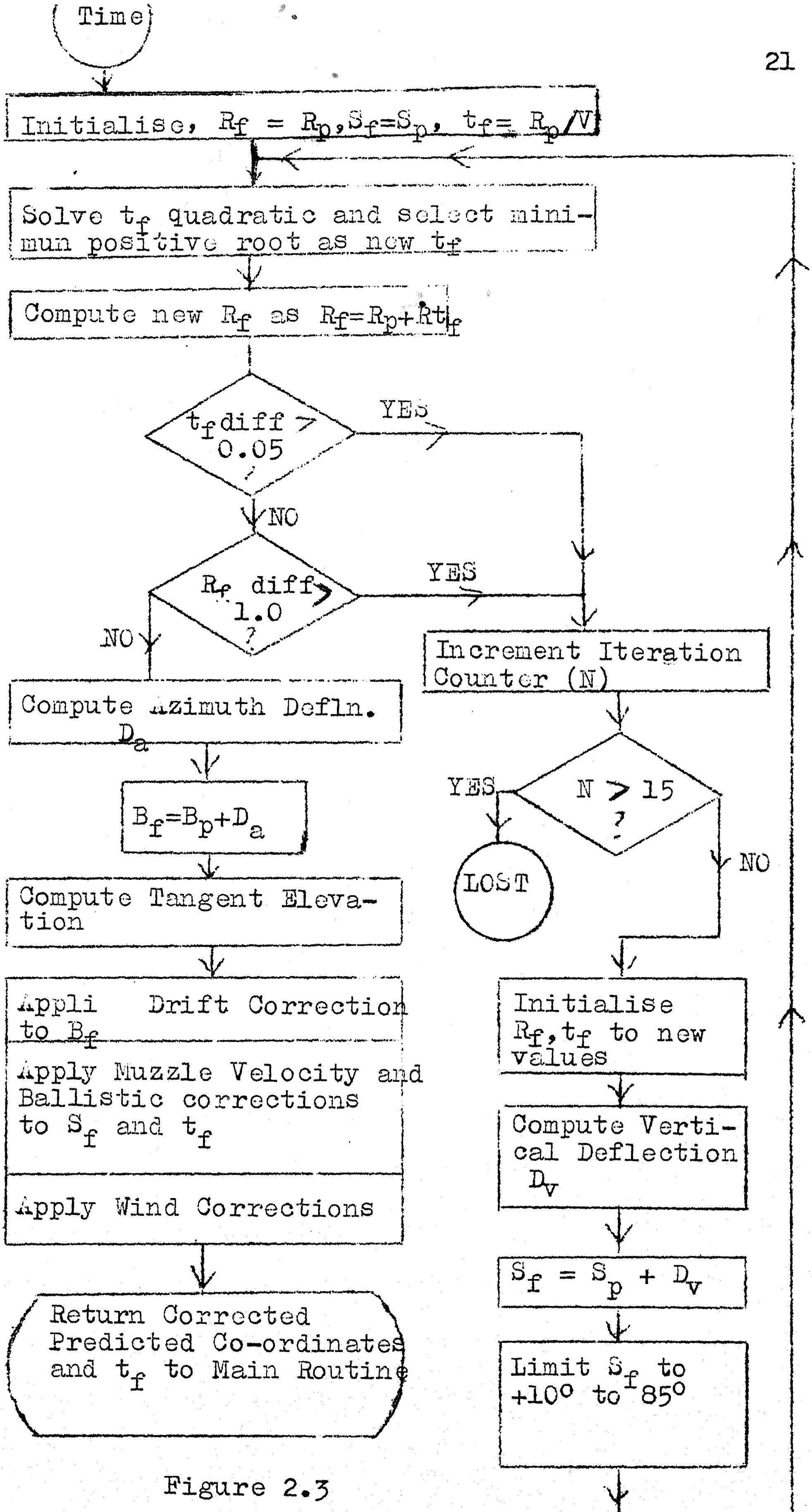


Figure 2.3

Time of Flight Flow-Chart

The time of flight is dependent upon the future range and angle of elevation. The elevation at which a gun can fire is generally limited between 10° and 80° as otherwise the ship's super-structures come in the way. To prevent a gun from firing outside the limits, mechanical switches which interrupt the firing circuit are incorporated in the guns. In the iterative calculations of time of flight an attempt is made to limit the gun elevation within 10° to 80° when the target is in the firing range.

In the design of the system there exists a time delay in determining the correction function coefficnets. This time delay is equal to the initial time of flight, when the target is first detected. To minimise this delay the correction function coefficients should be obtained earlier. This is achieved by assuming a hypothetical muzzle velocity which is much higher than the actual muzzle velocity. A muzzle velocity of ten times the muzzle velocity is used for this purpose. This is however, done when the target is detected at a long range ($\geq$ 20,000 yards), as it is then that the delay is large. The use of the correction coefficients determined with the hypothetical shell velocity, marginally effect the system performance [3]. Thus these values are used till the correction coefficients can be determined with actual muzzle velocity. When the range is

$\leq$ 15,000 yards the actual muzzle velocity is used for all computations. Thus within the firing range of the gun the correction coefficients computed with the actual muzzle velocity are always used.

The corrections discussed in the preceeding sections have been included in this routine. In the microprocessor implementation of this routine, a special algorithm to determine square roots in solving the time of flight quadratic is incorporated. This is discussed in detail in Chapter 5, Section 5.3.1.

3. IMPROVED PREDICTION TECHNIQUES

3.1 General Representation of the System

A simplified fire control system is shown in Figure 3.1. It is assumed that the system is stabilised i.e. the gun and the radar antenna are mounted on stabilised platforms and all data available is stabilised. This is achieved by applying correcting signals from the ship's gyro stabilisers. It is also assumed that the gun and the Radar antenna are theoretically at the same point and plane. This is achieved by applying Dip, Lateral and Vertical Convergence corrections to the angle at which the projectile is to be fired. (These terms are explained in appendix A.)

3.2 Predictor

Out of the four sub-systems constituting the fire control system, the one which is of interest to us is the Predictor. The inputs available to the Predictor are : (A) Through the Radar and TRACKER (device that generates the necessary signals for automatically tracking the target, once the radar is 'locked-on', to the target). These are :

1. Present range of the target (R_p)
2. Present bearing of the target (B_p)
3. Present elevation of the target (S_p)
4. Rate of change of range of the target ($\dot{R}$)
5. Rate of change of bearing of the target ($\dot{B}$)
6. Rate of change of elevation of the target ($\dot{S}$)

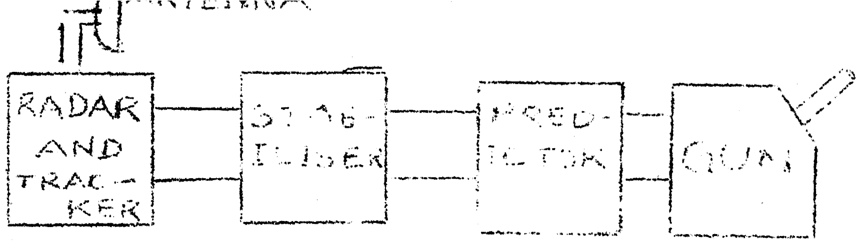


Figure 3.1

Simplified Fire Control System

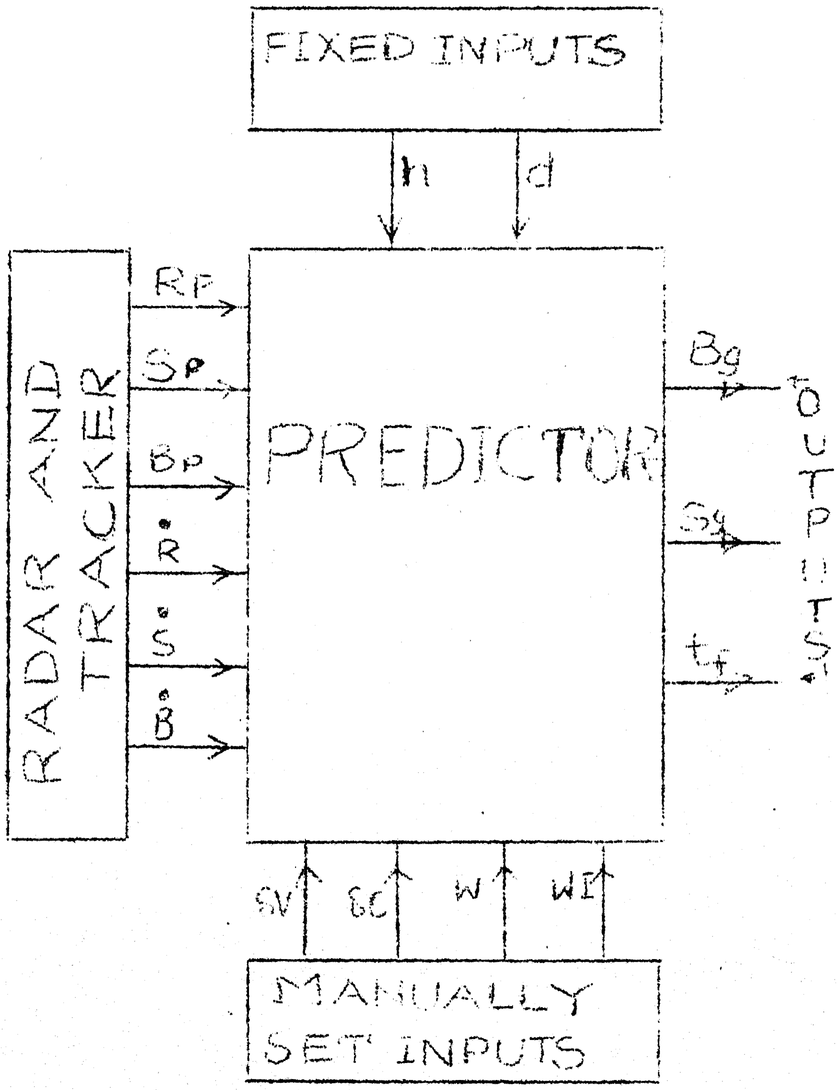


Fig. 3.2

Predictor Inputs and Outputs

(B) The set of the variable inputs that is manually fed to the predictor. These are :

1. Correction for change in Muzzle velocity (δV) - a factor determined from manuals particular to the gun and dependent on the number of rounds fired since the last calibration to the standard muzzle velocity [1].
2. Correction for change in Ballistic conditions (δC) - dependent upon atmospheric conditions.
3. Equivalent wind velocity, W
4. Wind Indination, WI.

(C) Set of fixed inputs for gun orders. These are :

1. Height difference between gun and director (h) - to compute DIP.
2. Horizontal distance between gun and director (d) - to compute lateral and vertical convergence.

The outputs from the predictor consist of :

1. Gun training (B_g)
2. Gun Elevation (S_g)
3. Time of Flight of Shell (t_f) - this is necessary for systems that employ time - set fuzes for shells, to set the fuze number.

These inputs and outputs are diagramatically shown in Fig. Figure 3.2.

The function of the predictor is to compute B_g, S_g and t_f from the available inputs. The equations used in the computations make use of some constants, which in real life situations are slowly varying variables. For simplifying the predictor design, these are assumed to be constants. Emperical relations derived after extensive trials are also used. Such imperfections introduce error in the system and reduce the accuracy of the system. The error can be divided into two parts :

(1) Error in prediction of future position of aircraft.
(2) Error in prediction of shell trajectory.

The latter error can be reduced by collecting sufficient data regarding shell trajectories through a number of experiments and trials under varying circumstances, and deducing emperical relations that can be applied as correcting functions at the time of firing. A correction technique to reduce the error in prediction of future position of aircraft is explained in the next section.

3.3 Correction Technique and Methods

The predicted future positions of the aircraft at regular time intervals is stored to find the error in prediction at

various time intervals. The error in prediction is determined by comparing the actual aircraft position with that of the predicted value after the elapse of the time of flight interval. If the error in prediction of the range is DR_{fi} when the predicted future range is R_{fi} for various i, then DR_{fi} is expressed as a function of R_{fi}. From a set of values of DR_{fi} a curve fit routine is used to compute the coefficients of this function. Once the coefficients are known, for any predicted future range R_{fx}, the error in prediction DR_{fx} is computed. This is applied as a correction to R_{fx} to reduce the error in prediction of R_{fx}. Similar logic applies to correction in elevation (DS_x) and bearing (DB_x).

Depending upon the behaviour pattern of the error values with respect to range, any of the following methods can be used as the curve fit routine to obtain the correction functions :

1. <u>No correction :</u> If the error values are always within the required tolerance, no correction is necessary. This condition is well nigh impossible.
2. <u>Constant correction :</u> If the error values are more or less constant (within tolerable limits) over a length of time, the predicted values are corrected by this constant to give the corrected predicted co-ordinates. This is the simplest method to implement in a microprocessor. However, for the cases studied this has been

found unsuitable as although the values are considerably constant over adjoining sampling intervals, the values drastically differ after a length of time and hence it is not suited for continuous tracking.

(c) Average correction : The average of a group of error values is used as the correction function. This is better than the constant correction method as the values change progressively and this method can be used for continuous tracking. This is also simple to implement in a microprocessor. Although, the errors by this method are brought within tolerances in a large number of cases, the method fails to reduce the errors in some cases. This method is suitable where the variation of error values between successive sampling intervals is small, or where the error values are large. Its oase of implementation in a micropro-crssor as compared to the curve fit method makes it a contender for consideration in a microprocessor system.

(d) Polynomial curve fitting : This is used in all cases where the variation of error with range is smooth, i.e. it can be represented by the form

$$DX = a_o + a_1 R_f + a_2 R_f^2 + \dots\dots\dots\dots$$

If the variation is linear a first order polynomial

$DX = a_o + a_1 R_f$ need only be considered.

This can however, be generalised to higher order polynomials. For the large number of cases studied the variation in error has been found to be mostly linear and the error in all cases is reduced to acceptable values. This thus, provides a more accurate technique . However, its implementation in a microprocessor system requires much greater memory and this takes more time for execution. This has been dealt with in details in Chapter 5 under the section on special algorithms.

(e) Dynamic prediction (adaptive): where the error

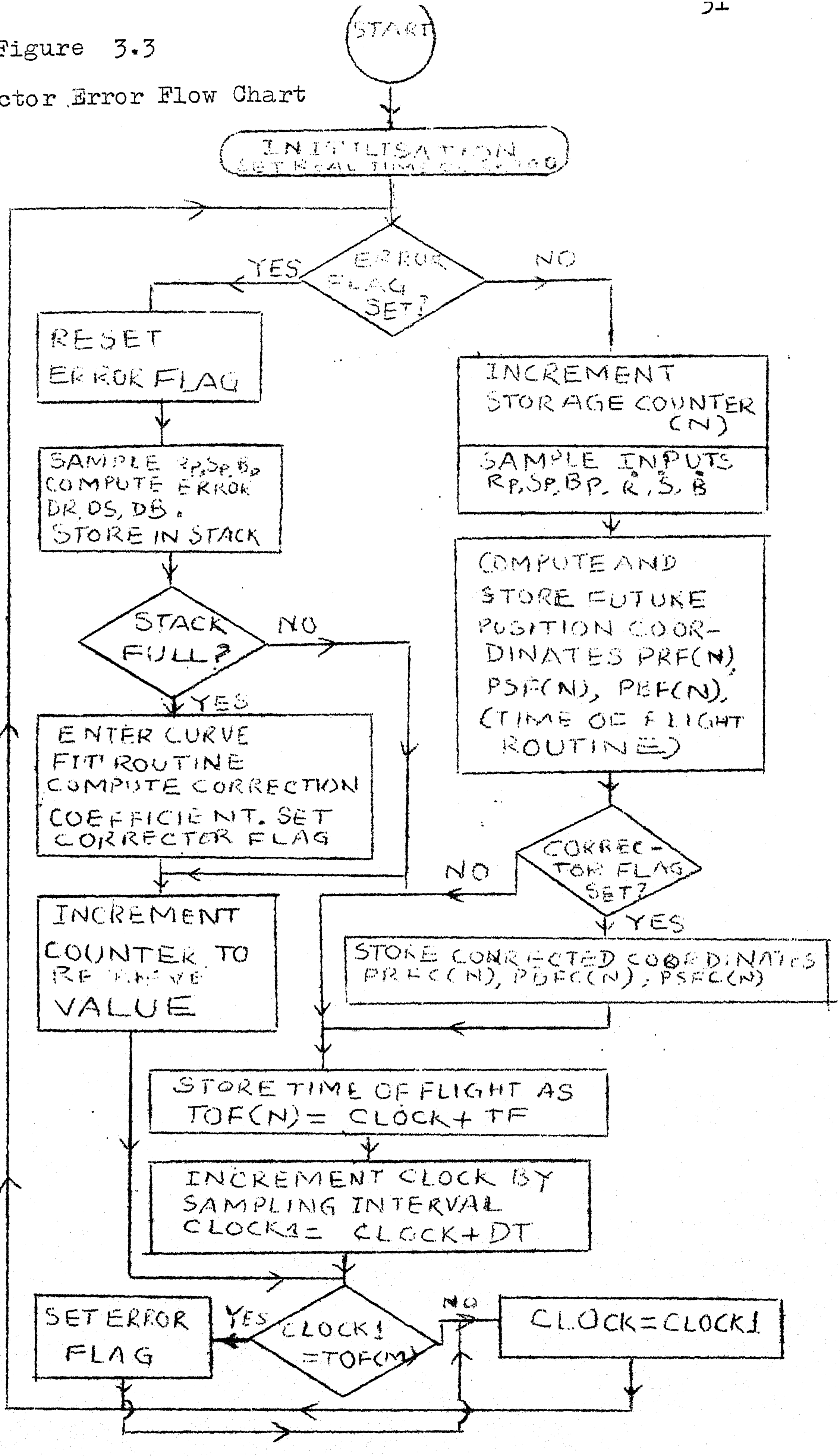
DX = f(range) where f is a function that minimizes some specified objective function as least-square error. f is in general a difficult function to determine and it may not be worth wasting the memory and execution time.

3.4 Prediction Error Routine

A flow chart indicating the computation and storing of Prediction Error is shown in Figure 3.3. When the predictor is switched ON or a fresh target is engaged, the system is initialised and the real time CLOCK is set to zero. It is assumed that the manual settings depending on the prevailing conditions are made prior to switching. The inputs from the radar and tracker are sampled and the predictor computes the future position of the aircraft and the time of flight of the shell incorporating all corrections as discussed in Chapter 2 (Time routine). The computed predicted co-ordinates are

Figure 3.3

Predictor Error Flow Chart



stored as PRF(1), PSF(1), PBF(1). The time of flight t_{f1} is added to the clock and stored as TOF(1). Now after the sampling interval dt , the inputs are sampled again and the future position and time of flight of shell, t_{f2} computed and stored as PRF(2), PBF(2), PSF(2). The CLOCK is added to t_{f2} and stored as TOF(2). This is repeated for every sampling time of 0.1 sec or 0.2 sec.

When the real time CLOCK equals TOF(1) the radar and tracker inputs are sampled to find the actual position of the aircraft as given by R_p, S_p and B_p. These are now compared (in the ERROR routine) with PRF(1), PSF(1), PBF(1) respectively to get the errors DR(1), DE(1) and DB(1). Again the inputs are sampled when the CLOCK equals TOF(2) to get the errors DR(2), DE(2) and DB(2) and this process continues.

A stack is set up for DR, DE, DB and the RANGE (R_f) for which these values are determined. A curve fit routine is entered which fits curves between the following pair of variables

1) DR vs R_f
2) DE vs R_f
3) DB vs R_f

The coefficients obtained therefrom are used to obtain the correction functions for range, elevation and bearing. Henceforth, for each input sampling, the future co-ordinates

PRF(N), PSF(N) and PBF(N) as well as their corrected values PRFC(N), PSFC(N) and PBFC(N) are computed and stored. If the aircraft is within the firing range of the gun, the firing can commence with the corrected values of the future position i.e. PRFC(N), PSFC(N) and PBFC(N).

Sample results of the cases studied indicating the variation of error in range, elevation and bearing without correction and with correction are given in Table 3.1(a) to (e).

The cases chosen correspond to :

a) Short range, fast approaching target .

b) Short range, fast receeding target .

c) Short range, slow approaching target.

d) Long range, fast approaching target.

e) Long range, fast receeding target.

Dimension for parameters are :

Range in feet, range rate in feet per second, elevation in degrees, elevation rate in degrees per second, bearing in degrees, bearing rate in degrees per second, wind velocity in feet per second, wind inclination in degrees, elevation and bearing errors in minutes, time in seconds.

Table 3.1(a)

Sample Data 1

Constant Values are W = 74.02063 WI = 21.29906

RRATE = -700.00 SRATE = 0.500 BRATE = 0.500

Initial Values are RP = 45000.00 SP = 45.00 BP = 135.00

Initial time of flight = 16.1822

TIME AFTER DETECTION		RANGE ERROR	ELEVATION ERROR	BEARING ERROR
28.3	U	530.32373	-116.60914	61.82303
	C	24.36743	-4.49531	-0.95028
31.9	U	459.77759	-113.13882	69.16962
	C	23.94946	-2.09795	0.25070
35.6	U	393.73047	-109.51945	76.86166
	C	21.44385	-0.94767	0.93338
39.2	U	332.40503	-106.51945	84.99652
	C	19.12573	-0.82155	1.26335
42.9	U	275.98096	-103.48161	93.74483
	C	16.87927	-0.64222	1.62036

Fired at target after 42.9 seconds of detection

Time of flight = 5.1635 seconds

U = Uncorrected Error

C = Corrected Error

Table 3.1(b)

Sample Data 2.

Constant values are W = 88.70099 WI = 23.89209

RRATE = 400.00 SRATE = -0.500 BRATE = 0.050

Initial values are RP = 21000.00 SP = 40.000 BP = 180.00

Initial Time of Flight = 11.8073

TIME AFTER DETECTION		RANGE ERROR	ELEVATION ERROR	BEARING ERROR
27.0	U	747.59446	-170.73746	49.70097
	C	5.70386	-4.38291	0.69804
33.0	U	827.81055	-185.84440	46.81563
	C	7.69238	-8.20768	1.01285
39.0	U	908.39844	-203.58168	44.11432
	C	7.66748	-12.29838	1.18869
44.9	U	988.87207	-224.77701	41.59079
	C	6.91016	-18.17242	1.38430
50.8	U	1068.72461	-250.54949	39.24096
	C	2.54883	-21.26823	1.12346

Fired at target after 50.8 seconds of detection.

Time of flight = 22.4108 seconds.

Table 3.1(c)

Sample Data 3

Constant Values are W = 89.63019 WI = 28.03121
RRATE = -200.00 SRATE = 1.000 BRATE = 1.000

Initial values are RP = 18000.000 SP = 45.000 BP = 260.00

Initial time of flight = 7.5805

TIME AFTER DETECTION		RANGE ERROR	ELEVATION ERROR	BEARING ERROR
14.9	U	292.85486	-136.33244	103.85501
	C	2.33337	0.00430	5.03087
19.4	U	240.74109	-129.66821	126.80995
	C	5.78442	-3.11157	11.14218
23.8	U	192.00696	-124.49663	158.66980
	C	5.79224	-2.52230	14.91912
28.3	U	146.97839	-119.95498	207.26841
	C	5.32153	-1.94387	22.34560
32.8	U	106.10266	-116.29331	291.49377
	C	4.50317	-0.98105	39.64620

Table 3.1(d)

Sample Data 4

Constant values are W = 88.98053 WI = 61.56277

RRATE = -600.00 SRATE = 2.000 BRATE = 2.000

Initial Values are RP = 90000.000 SP = 25.000 BP = 90.000

Initial Time of Flight = 3.9071

TIME AFTER DETECTION		RANGE ERROR	ELEVATION ERROR	BEARING ERROR
8.2	U	215.00098	14.59560	-11.43042
	C	-0.96289	-0.34180	0.21168
13.0	U	192.73145	16.09946	-10.33763
	C	-2.25098	-1.01016	0.69916
17.8	U	169.17676	17.08687	-8.13957
	C	-0.86523	-0.35363	0.90122
22.5	U	145.15625	17.74512	- 3.61605
	C	-0.10449	-0.14215	1.46479
27.3	U	121.45605	18.16956	7.30400
	C	0.34570	-0.21747	6.38219

Fired at target after 27.3 seconds of detection

Time of Flight = 3.2233 seconds (with increased muzzle velocity)

Table 3.1(e)

Sample Data 5.

Constant Values are W = 88.04145 WI = 58.61721

RRATE = 500.000 SRATE = -1.500 BRATE = -0.050

Initial Values are RP = 75000.00 SP = 45.000 BP = 1.000

Initial time of flight = 3.3225

TIME AFTER DETECTION		RANGE ERROR	ELEVATION ERROR	BEARING ERROR
7.2	U	98.03516	14.71017	13.20212
	C	-0.37012	1.26967	0.14789
12.2	U	112.26367	11.80898	12.18540
	C	-1.29980	-1.05926	0.39178
17.2	U	124.68555	6.47944	11.49490
	C	-1.14355	-1.76174	0.18448
22.2	U	134.93848	-5.84812	11.10838
	C	-2.39355	-7.52430	0.33099
27.2	U	142.69043	-61.10007	10.84027
	C	-2.02441	-40.62914	-0.10405

Fired at target after 27.2 seconds,

Time of Flight = 3.9124 seconds (with increased muzzle velocity)

4. SYSTEM DESIGN

4.1 Introduction to Microprocessor Architecture [6,7]

A review of the microprocessor is given in this section. A digital computer basically involves a control unit, an arithmetic and logical unit, and a randomly addressable memory. Instructions and data are stored in the memory. A microprocessor is an arithmetic and logical unit plus control unit realised on a small number of LSI chips. It provides the basic arithmetic and central processing unit of the computer. The instruction decoding, in microprocessors, may be done by random logic realized on the chip, or an alternative is to build a very elementary instruction set, called micro-instructions, whose functions are simply related to the hardware and then to realize each macro-instruction as a 'subroutine' made up of these microinstructions.

Byte Size

Most microprocessors have 8 bits for a byte. In most of the communication systems, display systems, one does not generally require more than 8 bit data and further 8 bits of data is a respectable level for A/D, D/A conversion. However, for complex arithmetic requirements microprocessors with 16 bits are available.

Registers

Registers correspond to writeable memory and so are especially important in microprocessor applications in which most memory is read only. 3 general uses of registers are

(i) for arithmetic and logical operations

(ii) for input/output and memory addressing operation

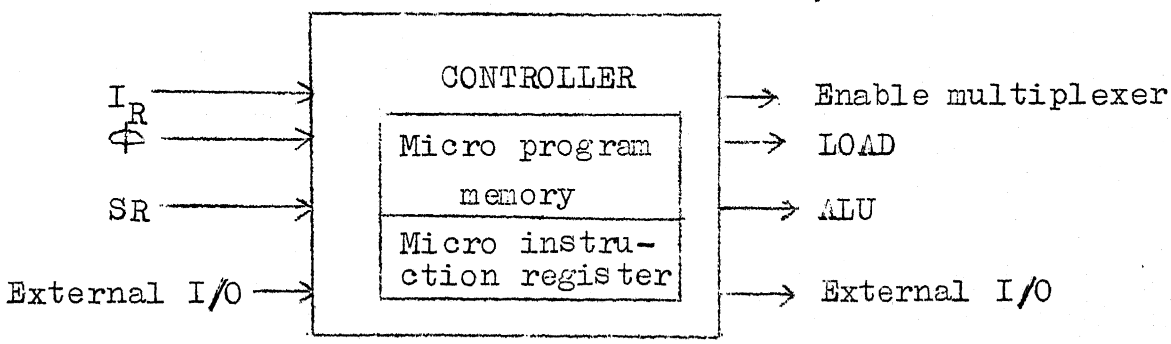
(iii) for maintaining control over the microprocessor operation itself (stacks, program counter and special purpose registers)

One or more accumulators or general purpose registers are normally provided and used both for temporary storage of data and also to provide source and destinations for operations, performed in the ALU. A large number of microprocessors also have an Index Register to facilitate indexed addressing. This can usually be incremented/decremented under program control. Some microprocessors are provided with only GPRs and these can be used either as accumulators or as index registers. Application programs are generally organized in subroutines and interrupt handling routines, the former to modularize the software and the latter to synchronize the microprocessor with the external inputs. Both lead to the need to interrupt normal processing sequence, process a routine, and then return to the point of interruption. For this, microprocessors provide some kind of a stack of

registers which facilitate the saving and restoring of the program counter. In some, a fixed set of registers in the microprocessor itself provide this facility (aided by special instructions for adding an address to the stack and deleting it later). In others, only pointers to the stack are provided in registers in the microprocessors, with the stack itself maintained in writeable memory.

Microprocessors also incorporate a status register which indicate the internal states of machine.

Most microprocessors use a microprogrammable controller. This is typically organised as shown diagrammatically in Figure 4.1 below.



4 different types of data come into it :

(i) information from instruction register (IR)

(ii) timing information (ϕ) (a) square wave output of clock
(b) external timing signals

(iii) internal states of machine, i.e. information from status register (SR)

(iv) external Input/output

The outputs from the controller consist of

(1) Enable multiplexers - this sets up data paths

(2) Load signals - this along with (1) above is required to enable registers to communicate amongst themselves.

(3) Signal to arithmetic and logic unit (ALU) to set up states

(4) External Input/output signals.

Bus Organisation

In a microprocessor system there is one bus in the system with n lines, and devices are 'hung on the bus'. A typical organisation is as shown in Figure 4.2.

An instruction fetch cycle would thus have the following sequence :

PC contains address of instruction

(i) BUS $\longleftarrow$ PC

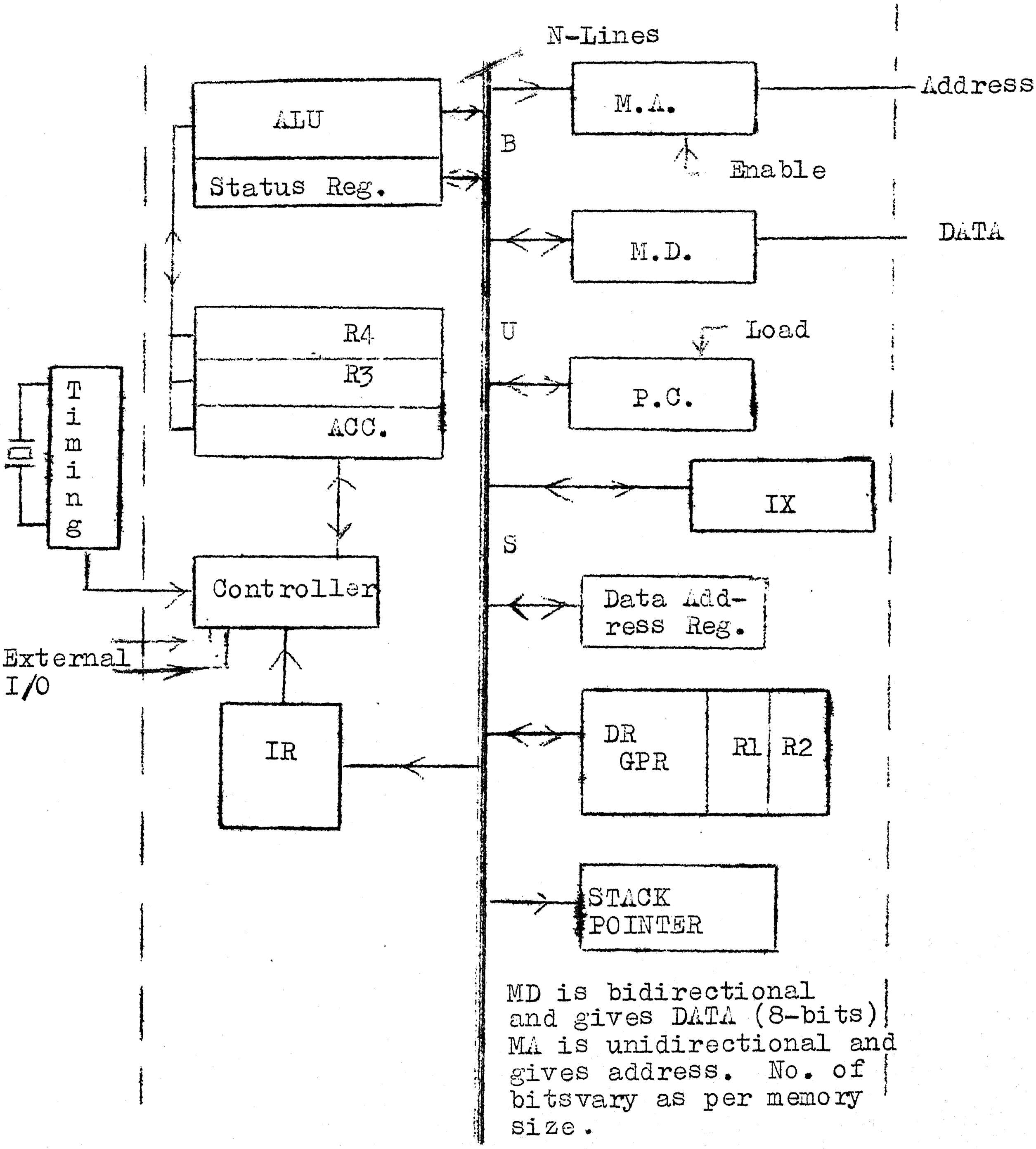
(ii) MA $\longleftarrow$ BUS

(waiting period - memory fetch)

(iii) BUS $\longleftarrow$ MD

(iv) IR $\longleftarrow$ BUS

Assuming two operand instructions, in dealing with any instruction the following 5 pieces of information is required.



MD is bidirectional and gives DATA (8-bits) MA is unidirectional and gives address. No. of bitsvary as per memory size.

Figure 4.2

Microprocessor Bus

(i) Instruction address (normally contents of PC)

(ii) Operand A

(iii) Operand B

(iv) Result

(v) Next instruction address

In most microprocessor systems operand A is in memory and operand B in some GPR (may be accumulator). Result is also put into some GPR and in simple microprocessor system this is also the accumulator. The next instruction address is in the Program Counter (the controller using information from Register/ALU loads the next address into the Program Counter).

Addressing Modes

The different addressing modes used in a microprocessor are

(a) Direct

(b) Extended

(c) Immediate

(d) Indexed

(e) Relative

One major addressing mode that is absent in microprocessors is the indirect addressing, that is the full address specified is the address of the address. However, microprocessors do have 'Register Indirect' addressing, where the register or register pair specified contains the address.

Interrupt System

The external interrupt is the main scheme by which external events and devices are synchronised with internal programs in a microprocessors. External interrupts signal the microprocessor that some event has occurred and the microprocessor then executes the application program associated with that event. The sequence of events in case of an interrupt is

(i) Store all information, i.e. save all the processor registers

(ii) Service the interrupt

(iii) Clear DONE

(iv) Restore all processor registers

(v) Reenable the interrupt, i.e. execute EI (enable interrupt)

(vi) Return to interrupted program

Memories :

Memories used with microprocessors could be of any technology depending on the speed and space requirement. The various processes are :

(i) N-MOS memories

(ii) P-MOS memories (occassionally - advantage is that voltage range can be greater)

(iii) CMOS - slowest but uses least amount of power

(iv) Bipolar - faster than NMOS by a factor of 10

- consumes much more power
- much larger in area and volume and therefore densities are not as great

(v) ECL - fastest (memories of order of nanoseconds)

The different types of memories that would be required depending on the applications and cost in mind are :

(i) ROM - read only memories - usually used for storing fixed programs. These are masked at the factory

(ii) FPROM-Field programmable read only memory
- cost per bit is 2 to 5 times that of ROM
- in this diodes are linked

(iii)EPROM- Electronically programmable ROM
- these can be erased by ultra violet radiation

(iv)EEPROM-Electronically erasable programmable ROM
- these can be erased electrically instead of ultra violet light

(V)EAPROM- Electrically alterable PROM
- these can selectively erase

All the above are non-volatile memories

(vi) RAM Random access memory
- memory which can be written or read

(vii)SSRAM Solid State RAM

4.2 Microprocessor Imposed Constraints

There are several important limitations of microprocessors which must be considered in the design of a microprocrssor based system. Any system that is designed without consideration of these limitations will result in a system which is, at best marginal in performance.

(a) Speed : Compared to either analog or discrete logic systems, microprocessors are slow. Gate time for standard TTL discrete circuits are measured in nanoseconds. Even the faster N-channel metal oxide semiconductor (NMOS) microprocessors are much slower. A typical memory to register add instruction will require 2 to 10 μseconds in most microprocessors.

(b) Short Word Length : Microprocessors typically operate on short word lengths. Most microprocessors utilise words of from 4 to 16 bits with 8 bits being most common. A word length of 8 bits is only sufficient for values upto 256. Thus in application, like ours, where angles upto 359.9 degrees, with resolution to the nearest tenth of a degree, and values of range upto 30,000 yards with accuracy of 1 yard, multiple word storage of data will be necessary. This will require additional time for computation.

(c) <u>Limited Arithmetic Capability :</u> Microprocessors have very limited arithmetic capability. There are currently no microprocessors with multiply or divide instructions. For applications that are not time bound one overcomes this handicap through software routines to multiply and divide. However, in real time applications, like ours, where the execution time is critical, hardware support in terms of multipliers and dividers is essential and is therefore suggested in the design.

4.3 Choice of Microprocessor

A very debatable point is the preference of a microprocrssor to a discrete logic system. In the systems under design a microprocessor was found to be essential for the following reasons :

(a) With a microprocessor there is a considerable flexibility in the choice of different parameters (like the setting of different coefficients). Also as there is flexibility in the choice of co-ordinate systems, use of microprocessor makes things simpler in transforming from any co-ordinate system to the desired co-ordinate system.

(b) The system requires data buffering in that the predicted co-ordinates are continuously stored and along with other changing, parameters are required for continuous computation.

(c) The microprocessor response is made fast enough with the inclusion of hardware modules for time consuming floating point arithmetic and trignometric functions.

The reasons for favouring Intel 8080 were the availability of product literature, software support, its ease of availability in the country, a rich instruction set and the fact that it is recognised as an industrial standard. A 16 bit microprocessors may perhaps have been better suited for the arithmetic but working with 4 bytes floating point arithmetic on 8080, with added hardware support is considered adequate to meet the timing constraints. An exhaustive evaluation procedure was not done because sufficient amount of literature about competitive processors was not available.

4.4 Description of Intel 8080

The Intel 8080 microprocessor is a 40 pin single-chip NMOS CPU which contains instruction decoding hardware on the same chip as the arithmetic and logic unit and is upward compatible with the Intel 8008 microprocessor. The 8080 has a 16-bit address bus, a 8-bit bidirectional data bus and fully decoded, TTL-compatible control outputs. It can support upto 64K bytes of mixed RAM and ROM memory, and can address upto 256 INPUT/OUTPUT ports. It has a basic cycle time of 2 μsec [5].

Architecture

The 8080 CPU consists of the following functional units [8]

(i) Register array and address logic

(ii) Arithmetic and logic unit (ALU)

(iii) Instruction Register and Control Section

(iv) Bi-directional, 3 state data bus buffer

A block diagram of the 8080 CPU is shown in Figure 4.3.

1. Register Array and Address Logic

The register section consists of a static RAM array organised into six 16-bit registers. These are (i) Program Counter (PC) which maintains the memory address of the current program instruction and increments upon every instruction fetch, (ii) Stack Pointer (SP) - maintains the address of the next stack location in memory, (iii) - (vi) Eight 8-bit registers arranged in pairs, referred to as B,C; D,E; H,L; and WZ. The temporary register pair W,Z is not program addressable and is only used for the internal execution of instructions. The other six general purpose registers can be used either as single registers (8-bit) or as register pairs (16 bit).

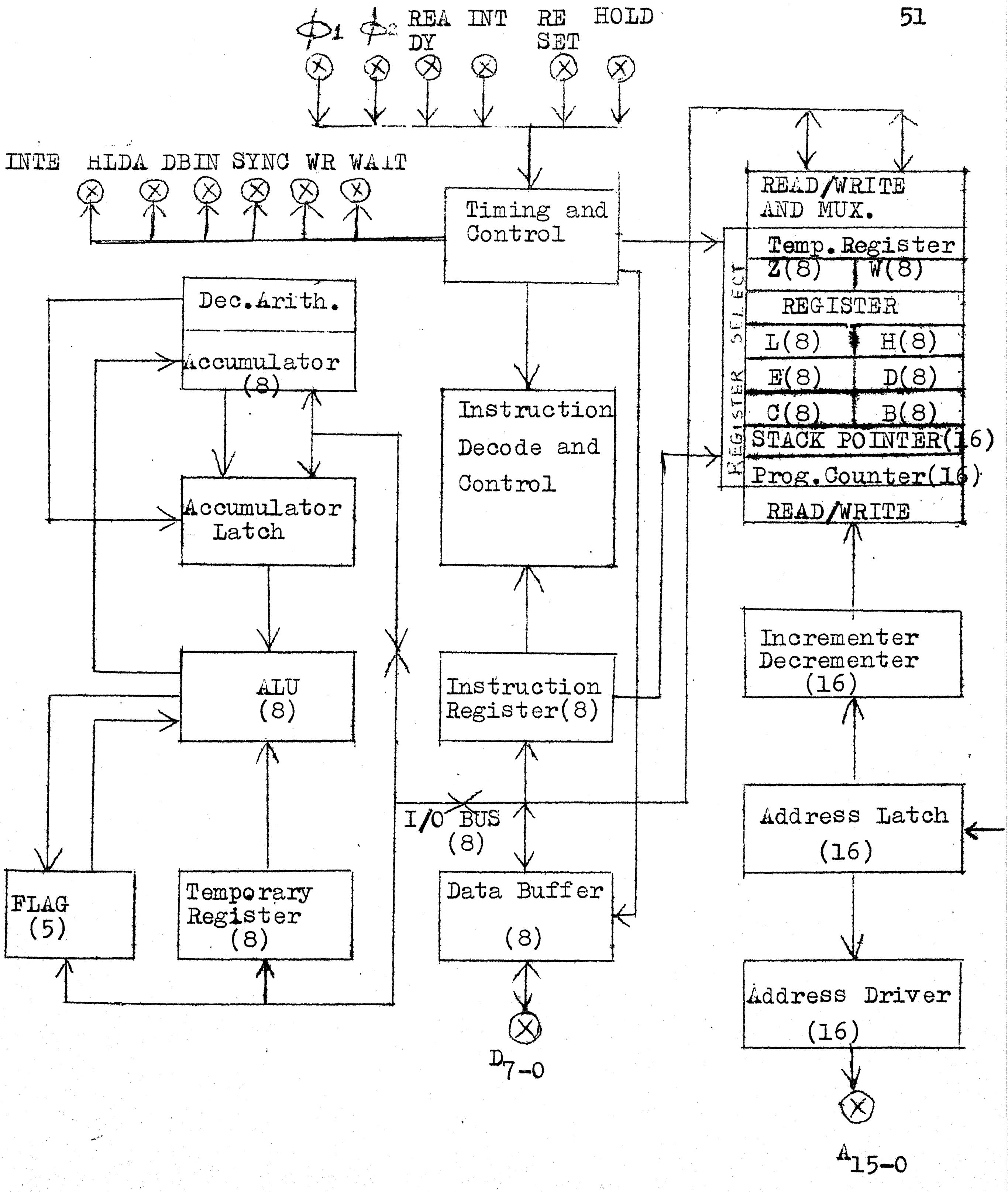


Figure 4.3

Block Diagram of the 8080

2. Arithmetic and Logic Unit

Arithmetic, logical and rotate operations are performed in the ALU. Registers which function with the ALU include an 8-bit accumulator, an 8-bit temporary accumulator (ACT), a 5-bit flag register (zero, carry, sign, parity and auxilliary carry) and an 8-bit temporary register (TMP). The ALU is fed by the temporary register (TMP) and the temporary accumulator (ACT) and carry flip flop. The result of the operation can be transferred to the internal bus or to the accumulator.

3. Instruction Register (IR) and Control Section

The IR is a 8-bit register which transfers information from the internal bus to the instruction decode and control section. The output of the decoder, combined with timing signals from the timing and state control section, provides the control signals for the memory, ALU, and data bus buffer. Outputs from the instruction decoder and external control signals feed the timing and state control section which generates the state and cycle timing signals. Inputs into the timing section include clock inputs ϕ_1 and ϕ_2 (the inputs of a two-phase external clock) and four internal control inputs READY, INT (Interrupt), RESET, and HOLD. Six direct control outputs provided for external control of memory and

I/O devices are INTE (Interrupt Enable), HLDA (Hold Acknowledge), DBIN (Data Bus IN), SYNC, $\overline{WR}$ (Write), and WAIT.

4. Data Bus Buffer

The data bus buffer is an 8-bit, bi-directional, tri-state buffer used to isolate the internal bus of the processor from the external data bus (D_0 - D_7). In the output mode, the content of the internal bus is transferred into an 8-bit latch. This latch drives the data bus output buffers. These buffers are switched off during input or non-transfer operations. In the input mode, data from the external data bus is transferred to the internal bus.

<u>Instruction Cycle</u>

INSTRUCTION FETCH, MEMORY READ, MEMORY WRITE, STACK READ, STACK WRITE, INTERRUPT, HALT, INPUT and OUTPUT are the possible machine cycles within the processor. No instruction cycle consists of more than five of these cycles. A machine cycle in the 8080 consists of three, four or five states, where each state is equal to the time interval of one period of the two-phase clock. Typically the clock frequency is 2 MHz, which results in a state of 0.5 µs duration. The five states are referred to as T1, T2, T3, T4 and T5. All machine cycles do not require the full five states, although none requires less than three. Hence a machine cycle varies from 1.5 to 2.5 micro seconds. The activities of the

different states are summarised in Table 4.1. States T1, T2, T3 are common to all machine cycles. If the processor has to wait for a response from an I/O device, then the cycle may contain one or more wait states TW. During these first three states, data is transferred to or from the processor.

Table 4.1 8080 STATE DEFINITIONS

State	Associated Activities
T1	A memory address or I/O device number is placed on the address bus; status information is placed on the data bus.
T2	The CPU samples the READY and HOLD inputs and checks for Halt instruction
TW (optional)	Processor enters WAIT state if READY is low or if Halt instruction has been executed.
T3	An instruction byte (FETCH cycle), data byte (MEMORY READ, STACK READ or INPUT CYCLE) or interrupt instruction (INTERRUPT cycle) is input to the CPU from the data bus; or a data byte (MEMORY WRITE, STACK WRITE, or OUTPUT cycle) is output onto the data bus.
T4,T5 (optional)	States T4 and T5 are available if the execution of a particular instruction requires them; if not, the CPU may skip one or both of them. T4 and T5 are only used for internal processor operations.

INSTRUCTION SET

The 8080 Instruction Set consists of 78 instructions. The instructions are either one, two or three bytes in length. Multiple byte instructions are stored in successive memory locations. The first byte gives the OP code, whereas the second and third bytes, if required, give the data or address. One byte instructions include register-to-register, memory reference, arithmetic, logical, Rotate, Return, Push Pop and Enable or Disable Interrupt Instructions. Two-byte instructions include immediate mode and I/O instructions. Three byte instructions include Jump, Call, Direct, Load and Store instructions. The instruction set can be broadly divided into four groups :

(a) Data Movement Group - This includes movement of data between registers, register to memory and memory to register. Eleven such instructions are available.

(b) Data Manipulation Group - This includes the instructions for performing arithmetic and logical operations on data. Twenty nine such instructions are there.

(c) Decision and control group - This group has twenty nine instructions which include branch instructions, unconditional and conditional (permitted for eight conditions) and for subroutines.

(d) Stack, I/O and Machine Control Group - Nine instructions for performing stack and I/O operations. Instructions for pushing and popping register pairs and status words to and from the stack are available.

Addressing Modes :

8080 has the following addressing modes :

(i) direct addressing

(ii) register addressing

(iii) register indirect

(iv) immediate addressing

It has no relative addressing.

4.5 System Requirements

A practical system should have a detection range from zero degrees to 359.999... degrees (that is 0 to 6.2834..... radians) in bearing and distance to atleast 30,000 yards. Assuming an accuracy of upto one minute in angular values and of one foot in distances, the system should be capable of representing values from 10^{-4} to 10^{5}. Intermediate calculations could approximately range from 10^{-6} to 10^{+6}. The ratio of the largest to the smallest number is about 10^{12}. If fixed point arithmetic is used approximately 4_1 bits words would be required. Since the range of numbers is large, floating point representation of numbers is indicated. A 6 digit decimal number would require

approximately 20 bits for a binary representation. Thus two bytes of 8080 would be insufficient to contain the number and hence 3 bytes (i.e. 24 bits) are used as the mantissa (normalised floating point). A 6 bit exponent is capable of handling numbers in the range of $2^{\pm 63}$ which is more than adequate to handle the range of numbers encountered during all computations inthe system. The arrangement adopted for representation of floating point numbers in the system therefore is to use 4 contigous bytes. The first byte is the exponent and the succeeding three bytes are the mantissa. The first bit of the first byte contains the sign of the number and the second bit the sign of the exponent. The mantissa is normalised and arranged that the most significant bits occupy the second byte with the most significant bit as the first bit of the byte and the least significant bit as the last bit of the fourth byte. Examples of 3 numbers in this representation are shown in Figure 4.4.

As no multiply, divide instructions are provided in 8080, software routines to do this are necessary. Software routines to carry out four byte floating point multiplication division, addition and subtraction take too much time in view of the large number of times these are required in the system. The system being time bound to the execution of the entire program in less than 100 ~~μsec.~~ milliseconds, hardware implementation of the above arithmetic functions is recommended.

D	E	H	L
0 0 0 0 1 1 1 0	1 0 0 1 1 1 0 0	0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0

(a)

D	E	H	L
1 0 0 0 0 1 1 0	1 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

(b)

D	E	H	L
1 1 0 0 0 0 1 0	1 1 0 0 1 1 0 0	1 1 0 0 1 1 0 0	1 1 0 0 1 1 0 0

(c)

Figure 4.4

Floating Point Representation of 3 Sample Numbers

(a) + 10000
(b) − 32.125
(c) − 0.20

This should have an execution time of less than 50 micro-second. For similar reasons hardware realisation of the trigonometric functions (sine, cosine and arc sine) is also recommended.

4.6 Proposed System Using 8080 and Other Hardware Module

A block schematic diagram of the proposed fire-control system using 8080, is presented in this section.

In our case, the three important considerations are as follows :

1. Floating point hardware module :

As the fire control problems involves a lot of floating point calculations, it is recommended that a floating point hardware is included. It interacts with CPU as follows. Since a floating points instruction is not a regular 8080 instruction, an illegal instruction trap will occur. In the trap handling routine, we will check whether it is a floating point-instruction or not. If so control is transferred to the floating point hardware. The floating point hardware, is capable of accessing the registers and the memory. (This is so because at the time the trap occurs, the execution of CPU is abandoned). Once the floating point hardware has completed the execution, it acknowledges to the control unit of CPU of its completion.

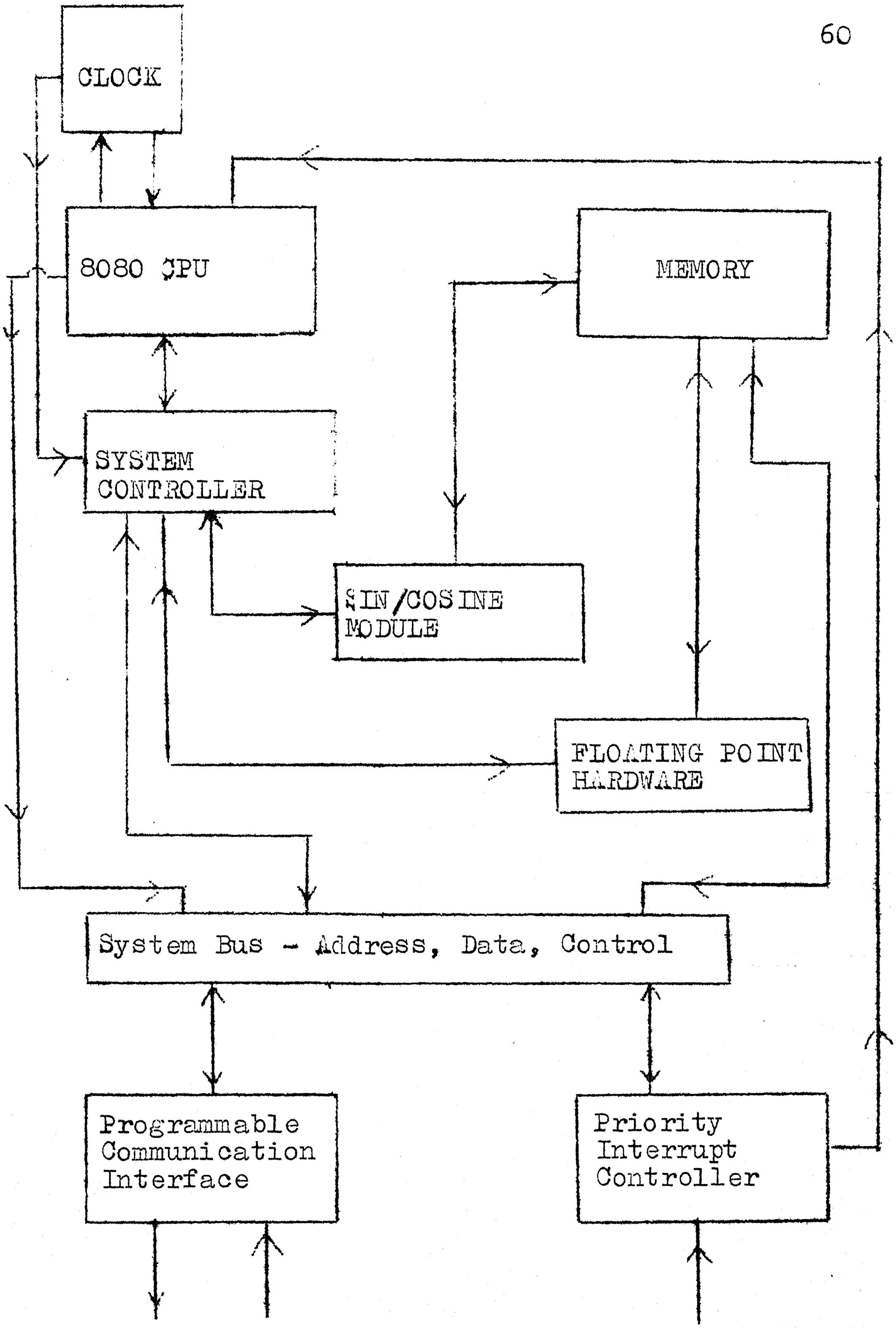


Figure 4.5

Block Schematic of the System

2. Sin/Cos/ARSIN Hardware :

We would also require the hardware for finding the SIN/COS/ARSIN values. Here there is a trade off between the execution speed (i.e. series expansion) and the memory space available (i.e. table loop-up). In either of the cases, the procedure similar to the one discussed above could be adopted.

3. Synchronization

We have assumed that the range (R_p), the elevation (S_p), and the bearing (B_p) are available continuously through an A/D converter as input in fixed memory locations. However, we do not want the values to change during our computational cycle. That is we want these memory locations to be locked to the input unit during the computation cycle and to be locked to the arithmetic unit during the input cycle. If we assume the input cycle to be one unit of indivisible instructions, then the case is considerably simplified.

In Figure 4.6 , whenever the execution cycle is progressing, the lock is set, so that the memory locations cannot be altered. But the A/D convertion process may go simultaneously during the execution cycle. Once the execution cycle is completed, the lock is reset and memory locations are altered by the input. The next execution cycle starts, only after 100 msec. from the start of the previous execution cycle. Once the execution cycle commences, the lock is set.

The output of the system (i.e. the gun orders comprising of Gun Training, Gun Elevation, and time of flight for fuze setting) goes directly to the gun setting mechanisms.

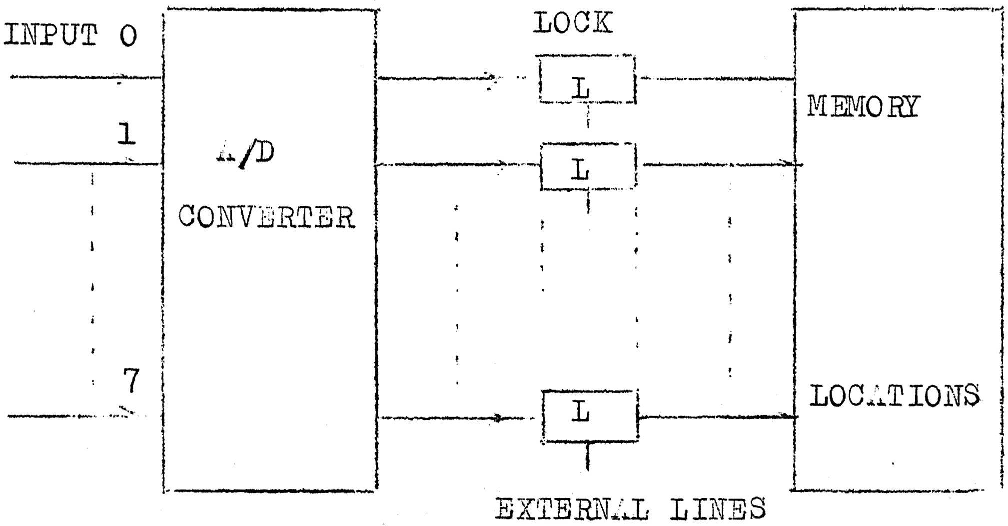


Fig. 4.4

Input - Memory Locking Arrangement

5. SYSTEM SOFTWARE

5.1 Discussion of 8080 Assembly Language with Particular Reference to Instructions Extensively Used

Assembly language programs are written as a sequence of instructions which are converted to executable hexadecimal code by a special program called an ASSEMBLER. The statement syntax of the 8080 assembly language is as follows :

Field 1 is, the LABEL field. This is an optional field, which if present, may be from 1 to 5 Characters long. The first character must be a letter of the alphabet or @ or? . A colon (:) must follow the last character.

Field 2 is the CODE field. This field contains a code which identifies the machine operation (add, jumps, subtract etc) to be performed. It contains one of the mnemonics which identify each instruction followed by atleast one space.

Field 3 is the OPERAND field. This provides any address or data information needed by the CODE field. Depending upon the code field, the operand field may be absent or may consist of one item or two items separated by a comma. The types of information that may be requested as items of an operand field and the ways in whichit may be specified is listed in Table 5.1.

Table 5.1

Operand Field Information

OPERAND FIELD INFORMATION	
Information required	Ways of specifying
(a) Register	(1) Hexadecimal data
(b) Register pair	(2) Decimal data
(c) Immediate data	(3) Octal data
(d) 16-bit memory address	(4) Binary data
	(5) Location counter ($)
	(6) ASCII constant
	(7) Labels assigned values
	(8) Labels of instructions
	(9) Expressions

Field 4 is the COMMENT field. This is present for the programmer's convenience and is ignored by the assembler. This must begin with a semicolon (;).

Instructions Extensively Used

A large number of the 78 instructions in the Instruction Set have not been used. Only 20 of these instructions have been used and these are listed and explained below. Many instructions from the Data Manipulation Group could not be

used as the system uses 4 byte floating point arithmetic. Extensive use has been made of the LHLD (Load Direct Register Pair H,L) and SHLD (Store Direct Register Pair H,L) instructions to manipulate 4 byte numbers. Instructions that have been used are

(A) FOR PRIMARY MEMORY REFERENCE

1. LHLD Load H and L Direct

   (L) $\leftarrow$ (address)

   (H) $\leftarrow$ (address + 1)

2. SHLD store H and L Direct

   (address) $\leftarrow$ (L)

   (address + 1) $\leftarrow$ (H)

3. LDA Load Accumulator Direct

   (A) $\leftarrow$ (address)

4. STA Store Accumulator Direct

   (address) $\leftarrow$ (A)

5. MOV Move Instruction

   The format is MOV dst, Src

   A,B,C,D,E,H,L or M

   (dst or src not both = M)

B. <u>FOR MEMORY REGISTER OPERATE</u>

6. Logical Exclusive - or Register or Memory with Accumulator (zero Accumulator)

This has been used to zero the accumulator since any bit exclusive-or ed with itself produces zero

$(A) \longleftarrow (A) \quad (+) \quad (A)$

7. CMP Compare Register or Memory with Accumulator

The format of this instruction is

CMP reg

The specified byte is compared to the contents of the accumulator. The comparison is formed by internally subtracting the contents of REG from the accumulator (leaving both unchanged) and setting the condition bits according to the result. The zero bit is set to 1 if $(A) = (r)$. The carry flag is set to 1 if $(A) < (r)$

8. INR Increment Register or Memory

Format INR reg (B,C,D,E,H,L,M or A)

INR A

$(A) \longleftarrow (A) + 1$

9. DCX Decrement Register Pair

Format DCX rp (B,D,H, or SP)

The 16-bit number held in the specified register pair is decremented by one.

10. XCHG Exchange Registers

(D) $\longleftrightarrow$ (H)

(E) $\longleftrightarrow$ (L)

The 16-bits of data held in the H and L registers are exchanged with the 16-bits of data held in the D and E registers.

11. SPHL Load SP from H and L

(SP) $\longleftarrow$ (H,L)

The 16-bits of data held in the H and L registers replace the contents of the stack pointer SP. The contents of H and L registers are unchanged.

12. MVI Move Immediate Data

Format MVI R, DATA

(R) $\longleftarrow$ DATA (1 byte data)

13. LXI Load Register Pair Immediate

Format LXI RP, DATA (2 bytes data)

The most significant 8 bits of the 16-bit immediate data is loaded into the first register of the register pair, while the least significant 8 bits are loaded into the next register of the register pair.

C. <u>FOR BRANCHING</u>

14. JMP Jump

Format JMP ADDR

(PC) $\longleftarrow$ ADDR

Jump to instruction with label ADDR

15. JM Jump if Minus

Jump if sign bit = 1

16. JP Jump if positive

Jump if sign bit = 0

17. JNZ Jump if not zero

Jump if zero bit = 0

18. JZ Jump if zero

Jump if zero bit = 1

D. FOR STACK OPERATION

19. PUSH Push Data onto Stack

Format PUSH rp (B,D,H, or PSW)

The contents of the register pair are saved in two bytes of memory indicated by the stack pointer SP.

(SP-1) ← (First register)

(SP-2) ← (Second register)

SP ← SP - 2

20. POP Pop Data Off Stack

Format POP rp (B,D,H, or PSW)

The contents of the register pair are restored from two bytes of memory indicated by the stack pointer SP.

(Second register) ← (SP)

(First register) ← (SP + 1)

SP ← SP + 2

E. INSTRUCTIONS ADDED AS A RESULT OF ADDITIONAL HARDWARE

1. FADD Floating Point 4 byte addition

Format FADD D, ADDR or

FADD ADDR, D

The 32 bit data held in register pairs D and H arranged as sign bit plus signed exponent in register D, most significant 8 bits of mantissa in register E, next 8 bits in H and least significant 8 bits of mantissa in register L, are added to the address, where ADDR contains the exponent and ADDR + 3 the least significant 8 bits of the mantissa. Thisfloating point addition is done by the additional hardware. The result of the addition is in registers D,E,H,L in the form as explained above.

2. FSUB Floating Point 4 byte subtraction

Format FSUB D, ADDR or

FSUB ADDR, D

In this case the first operand is treated as the minuend (i.e. the number to be subtracted)

3. FMUL Floating Point 4 byte multiplication

Format FMUL D, ADDR or

FMUL ADDR, D

4. FDIV Floating Point 4 bytedivision

Format FDIV D, ADDR or

FDIV ADDR, D

In this case the first operand is treated as the divisor.

5. COS Cosine

Format COS ----

The Cosine of the floating point number in the registers D,E,H,L is returned in the registers D,E,H,L in the form as described in 1.

6. SIN Sine

Format SIN -------

The sine of the floating point number in the registers D,E,H,L is returned in the registers

D,E,H,L

7. ARSIN Arc Sine

FORMAT ARSIN ------

The Arc Sine of the floating point number in the registers D,E,H,L is returned in the registers D,E,H,L.

5.2 MACROS

A macro is a means of specifying to the assembler that a symbol (the macro name) appearing in the code field of a statement actually stands for a group of instructions. The use of macros increases the efficiency of programming and

the readability of programs. Macros have been used while programming the system. Instead of defining these within the main program, all the macros used have been defined before the start of the main program and these have only been referenced or called in the text of the program.

The format for the Macro Definition is

Label	Code	Operand
name	MACRO	Plist
macro	body	
	ENDM	

'name' is the name of the macro
'plist' is a list of unquoted character strings identifying the dummy parameters appearing in the body of the macro-definition. Subsequent macro references or call specifying the actual parameters to be substituted for the dummy parameters must adhere to the positionality of the parameters as indicated in 'plist'.

5.3 Special Algorithms

Algorithms used for working with microprocessors should be simple, so as to occupy a small portion of memory and should take very little time so that the main program is not time-bound to the execution of a particular algorithm. In the design of this system we have made use of two special algorithms that are simple and quick to execute.

5.3.1 Finding Square Root of a Positive Number

Finding the square-root of a positive number A is equivalent to finding the positive solution of the equation $f(x) = x^2 - A = 0$. Probably the best-known of all iterative methods to solve this is that of Newton Raphson, where you generate successive approximations from the iteration.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad \text{i.e. for}$$

the case under consideration

$$x_{i+1} = x_i - \frac{x_i^2 - A}{2\,x_i} = (x_i + \frac{A}{x_i})/2$$

The success and rapid convergence of this method is dependent upon the choice of the initial guess. If the initial choice happens to be near the root the technique converges very fast.

For the system under consideration, this routine is required only for solving the quadratic that determines the time of flight.

The quadratic to determine time of flight is

$$R_f = Vt\ (1 - \frac{gt}{2V} \text{ Sin } S_f)$$

i.e. $g \text{ Sin } S_f\ t^2 - 2Vt + 2R_f = 0$

$$\text{Therefore, } t = \frac{2V \pm \sqrt{4V^2 - 4.(g \text{ Sin } S_f).\ 2R_f}}{2\ g \text{ Sin } S_f}$$

The two roots, X1, X2 can then be written as

$$X1 = TERM + SQRT/2g \text{ Sin } S_f$$

$$X2 = TERM - SQRT/2g \text{ Sin } S_f$$

where $TERM = \frac{2V}{2 g \text{ Sin } S_f} = \frac{V}{g \text{ Sin } S_f}$

and $SQRT = \sqrt{4 V^2 - 8 R_f (g \text{ Sin } S_f)}$

When considering the time of flight we consider the minimum positive time, therefore, if SQRT is less than TERM we consider the root X2 if SQRT is greater than or equal to TERM we consider the root X1

if $SQRT < TERM$

$$SQRT = \left(\frac{V}{g \text{ Sin } S_f} - X2\right) * 2 G \text{ Sin } S_f$$

if $SQRT \geq TERM$

$$SQRT = \left(X1 - \frac{V}{g \text{ Sin } S_f}\right) * 2 G \text{ Sin } S_f$$

A near approximation for the time of flight is simply the ratio of the present range to the muzzle velocity, i.e., X1 or X2 can be approximate to R_p/V. Therefore, if R_p/V is greater than or equal to $\frac{V}{g \text{ Sin } S_f}$ the root corresponds to X1 (TERM + SQRT/2g Sin S_f) and if R_p/V is less than $\frac{V}{g \text{ Sin } S_f}$, the root corresponds to X2 (TERM - SQRT/2g Sin S_f). Accordingly the initial guess for determining the root is

$$X = \left(\frac{V}{g \text{ Sin } S_f} - \frac{R_p}{V}\right) 2 \text{ g Sin } S_f, \text{ and if}$$

$$\frac{R_p}{V} \geq \frac{V}{g \text{ Sin } S_f} \; ; \qquad X = -X$$

Implementing this, it has been found that upto an accuracy of 10^{-6} the square root can be determined in $\leq$ 5 iterations.

5.3.2 Curve Fit Routine

Different curve-fit routines for the general purpose computers have been available [10]. The theory of curve-fit routines has also been widely studied in the past. In the present context we will develop curve fit algorithm that is suitable for implementing in our microprocessor fire control system. Instead of going for a more general algorithm we attempt to develop a specific algorithm for the reasons as stated in the beginning of this chapter. So, instead of an elaborate, general curve-fit algorithm, we write a specific curve-fit algorithm to suit our specific goals.

Input : Input consists of a set of observations namely the aircraft range error, or bearing error, or elevation error for a set of future range.

Output : Coefficient of the polynomial, which fits closely the calculated errors.

From the results of a large number of cases of aircraft movements studied, a first order polynomial is sufficient to describe the different errors in terms of the future range. This can however, be generalised to a polynomial of atmost 3rd order if practical observations so warrant it.

Algorithm : Under least square estimation, we have to minimise $\sum_{j=1}^{n} (y_i - \sum a_i x_j^i)^2$, where y_j's are errors, and the x_j's are parameters(predicted range at which the error occurs.)

The minimum of the above function is 0, and this occurs when,

$$y_j - \sum a_i x_j^i = 0$$

Considering a second order polynomial

$$y_j = \sum_{i=0}^{2} a_i x_j^i \ , \ j = 1,2,3$$

i.e.

$$y_1 = a_o + a_1 x_1 + a_2 x_1^2$$

$$y_2 = a_o + a_1 x_2 + a_2 x_2^2$$

$$y_3 = a_o + a_1 x_3 + a_3 x_3^2$$

The constants a_o, a_1, a_2 are split into 3 different sets of constants corresponding to y_1, y_2, y_3 and then we take the average of these to get the final constant. Though one might raise serious questions it has been verified that this gives 'sufficiently' correct results.

For a first order polynomial this reduces to

$$y_1 = a_o + a_1 x_1$$

$$y_2 = a_o + a_1 x_2$$

i.e. $$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \begin{bmatrix} a_o \\ a_1 \end{bmatrix}$$

$$\begin{bmatrix} a_o \\ a_1 \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$= \frac{\begin{bmatrix} x_2 & -x_1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}{x_2 - x_1}$$

$$a_o = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}$$

$$a_1 = \frac{-y_1 + y_2}{x_2 - x_1}$$

From a set of six values of errors for six future ranges, three such values of a_o and a_1 are computed and then the average of these values is taken to get the correction function, i.e.

$$\text{Error} = A_o + A_1 * \text{Range}$$

This routine is repeated for calculating the error function for range, bearing and elevation. This simplified curve fit routine requires 768 bytes of memory storage and has an

execution speed of 5.115 msecs, assuming a time period of 500 ns.

A further simplified program to determine the correction function is simply averaging the error values. The average of the first three values is averaged to the next two set of three values each and the final result is used as the correction function. Although, this is not as accurate as the above curve fit routine this provides a reasonably high improvement factor in error correction at higher ranges and the improvement factor considerably reduces at lower ranges. In some cases where the error is marginal, the corrected and uncorrected errors are almost identical in this case. However, this has the advantage of occupying very little memory space 348 bytes(as compared to 768 bytes in the previous case) and very fast execution time (2.659 msecs as compared to 5.115 msec in the previous case).

In our implementation, the first-order polynomial curve fit is preferred to the averaging case because of the following reasons.

(a) The corrected error in the first order polynomial curve fit is smaller than the corresponding error in the averaging case, at all ranges.

(b) The execution time is not a bottleneck in our case, because of the appropriate choice of the sampling interval.

5.4 Memory Requirement

A major portion of the memory is required for the storing of the predicted co-ordinates; these being the future range, bearing and elevation and the time of flight. The factors that affect the size of stack for each of these are the maximum time of flight of shell and the sampling interval. For the cases studied, the maximum time of flight encountered was 23.7 seconds, however, a slower muzzle velocity or bad weather conditions could, hypothetically, increase the time of flight to about 40 seconds. Assuming a sampling interval of 0.1 second, 400 values of each predicted co-ordinates would require storage. All succeeding values computed are over-written on the previous values as a value once used for comparison is not required again. As each value takes 4 bytes of memory, 1600 bytes arerequired for each predicted value. Thus a total of 6400 bytes are required for the storage of the predicted co-ordinates. This requirement can be halved by reducing the sampling interval to 0.2 seconds.

Program storage requires 768 bytes for the Main Program, 768 bytes for the curve fitting routine and 1536 bytes for the time of flight routine. 250 bytes are required for storing program referenced addressesses and 168 bytes for storing the constants and other values in fixed memory locations. 24 bytes each are required for maintaining the

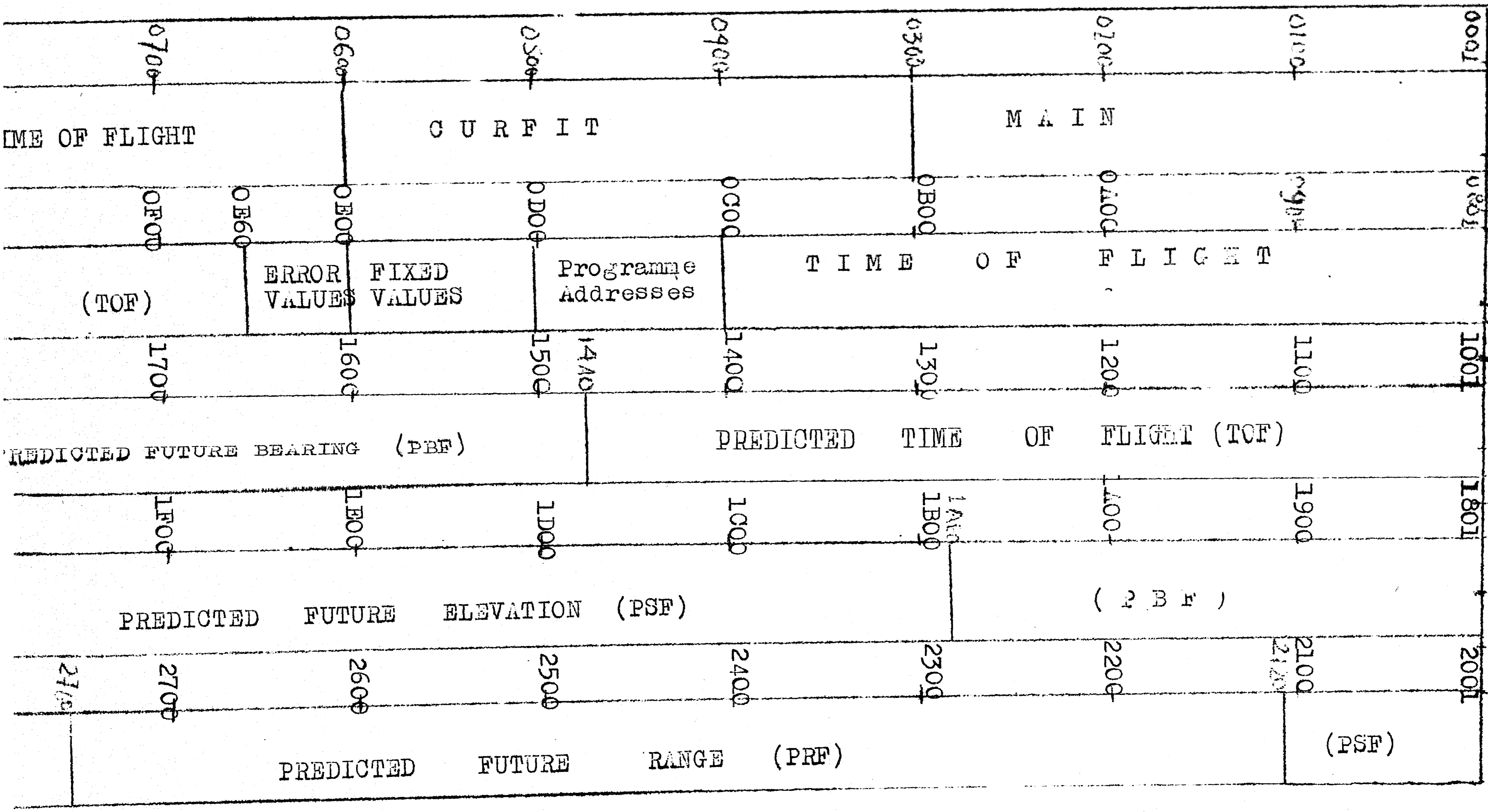
stacks for range error, bearing earor and elevation error. This total requirements addsupto 3586 bytes. Therefore, the total memory requirement works out to 9986 bytes. Hence a memory size of 10K bytes (10240 bytes) is recommended. A suggested memory map is given in Figure 5.1 and Figure 5.2 shows the details of the addresses occupied by the fixed values and the errors.

5.5 Timing Constraints

The number of time states associated with every 8080 CPU instruction used in our program is listed in Table 5.2. Instruction execution time equals number of time periods multiplied by the duration of a time period. A time period may vary from 480 nanosecs to 2 μsec, depending upon the memory type that is used. For computing the execution time of our program a time period of 500 nanosecs has been assumed. As the sampling interval chosen is 100 msec (0.1 sec), the number of time periods available for the execution of the entire program is 200,000.

The breakdown of the time periods required for the major subdivisions of the program are listed below :

Initialisation	479
Wind components computation	544
TIME routine assuming 10 iterations to compute time of flight and 5 iterations to compute squareroot of a number	120102



0001
0100
0200
0300
0400
0500
0600
0700
MAIN
CURFIT
IME OF FLIGHT
0801
0900
0A00
0B00
0C00
0D00
0E00
0E60
0F00
TIME OF FLIGHT
Programme Addresses
FIXED VALUES
ERROR VALUES
(TOF)
1001
1100
1200
1300
1400
14A0
1500
1600
1700
PREDICTED TIME OF FLIGHT (TOF)
REDICTED FUTURE BEARING (PBF)
1801
1900
1A00
1AA0
1B00
1C00
1D00
1E00
1F00
(PBF)
PREDICTED FUTURE ELEVATION (PSF)
2001
2100
2120
2200
2300
2400
2500
2600
2700
27A0
(PSF)
PREDICTED FUTURE RANGE (PRF)

Address	Contents
0D05	S_P
0D09	B_P
0D0D	$\dot{R}$
0D11	$\dot{S}$
0D15	$\dot{B}$
0D19	WI
0D1D	W
21	DRICON
25	V1
29	δV
2D	δC
31	C1
35	C2
39	C3
3D	C4
41	K
45	0·05
49	0·10
4D	2π
51	π/2
55	G=32
59	10°
5D	80°
61	89°
65	5°
69	70°
6D	0·0
71	1·0
75	2·0
79	−2·0
7D	3·0
0D80	

Address	Contents
	[illegible]
85	6·0
89	10·0
8D	30000·0
91	45000·0
95	60000·0
99	DT
9D	←
0DA1	

Address	Contents
0E01	RANGE
0E18	STAC4
	DB
0E30	STAC3
	DE
0E48	STAC2
	DR
0E60	STAC1
0E80	

Figure 5.2

Stack loading	504
Correction routine	816
Co-ordinates correction	516
t_f storage and clock increment	277
clock = t_f check	401
Error computation	861
Stack initialisation	137
Curve fitting routine using first order polynomial for curve fitting	10 230
Retrieval pointer initialisation	301
Clock = t_f check	445
Miscelleneous (counter initiation, flag check etc)	237
TOTAL	135850

The above figures correspond to maximum time periods for each subdivision. This corresponds to a maximum execution time of 67.925 millisecs. at start and a maximum of 67 millisecs once started (not including initialisation, wind components computation and initialisation of time of flight routine). This is well within the 100 milliseconds sampling interval, i.e. the interval after which the next values of R_p, S_p, B_p are available.

Table 5.2 INSTRUCTION EXECUTION TIMES

MNEMONIC	NUMBER OF TIME PERIODS
LXI	10
PUSH	11
POP	10
STA	13
LDA	13
XCHG	13
SPHL	4
INX	5
MOV r_1, r_2	5
MVI r,	5
INR	7
XRA	5
CMP	4
JMP	4
JZ	10
JNZ	10
JP	10
JM	10
DCX	5
SHLD	16
LHLD	16
FADD	100

contd... Table 5.2

MNEMONIC	NUMBER OF TIME PERIODS
FSUB	100
FMUL	100
FDIV	100
SIN	100
COS	100
ARSIN	100

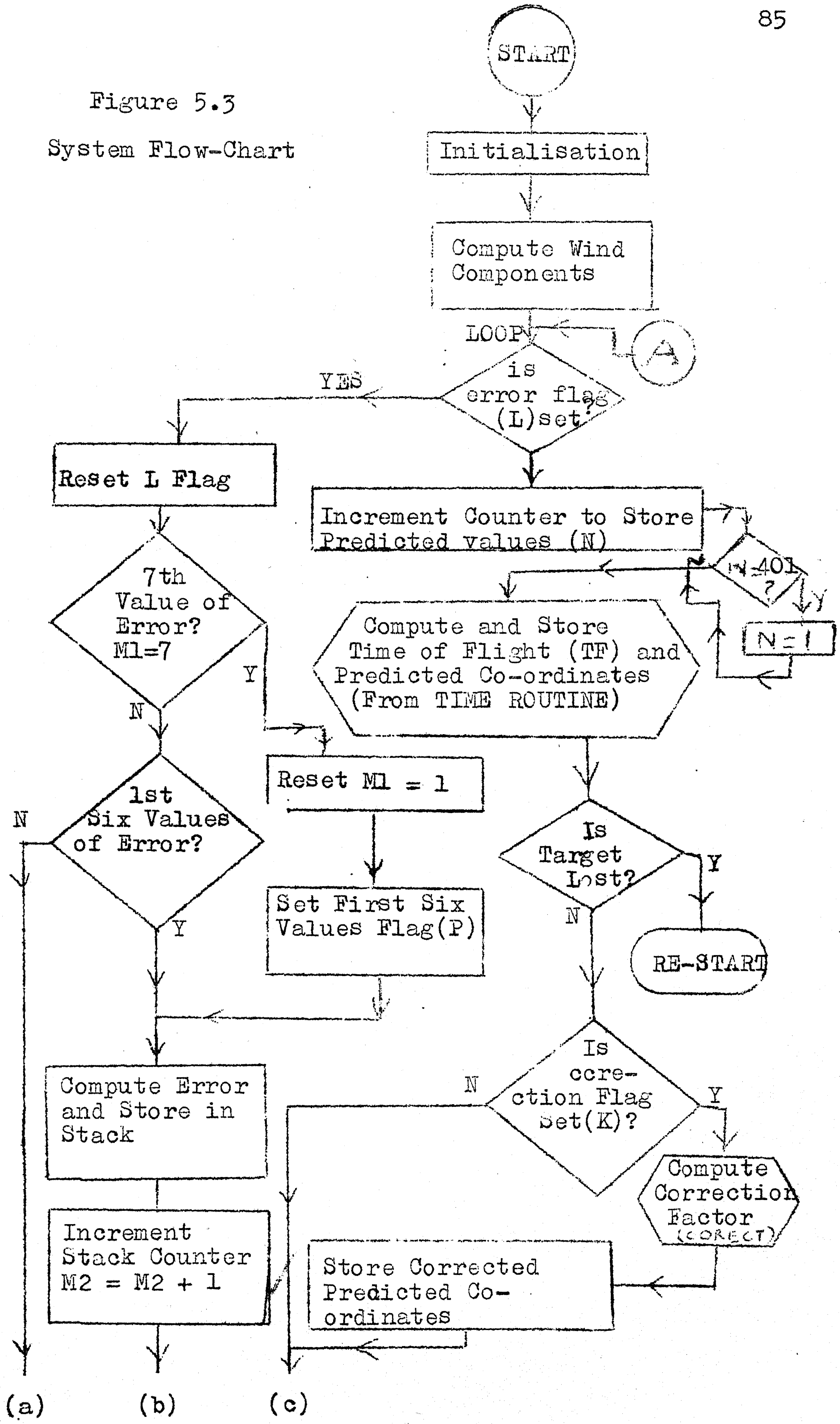
5.6 System Flowchart

The complete system flow chart is presented in Figure 5.3. The 8080 Assembly Language program corresponding to this is listed in Appendix C. It is assumed that the settings for the manual inputs are made prior to starting. It is further assumed that the wind conditions remain constant.

The time of flight and predicted co-ordinates are obtained from the time of flight routine. This has been dealt with in details in Section 2.5. These predicted co-ordinates are stored in separate stacks, the pointers for which are initialised at the start of the program. A maximum of 400 values can be stored. The 401 st. value is stored in the 1st location of the particular stack. Thus the system will work for maximum initial time of flight of 40 seconds. The tolerance for checking whether the predicted time of flight and clock values are same is restricted to half

Figure 5.3

System Flow-Chart



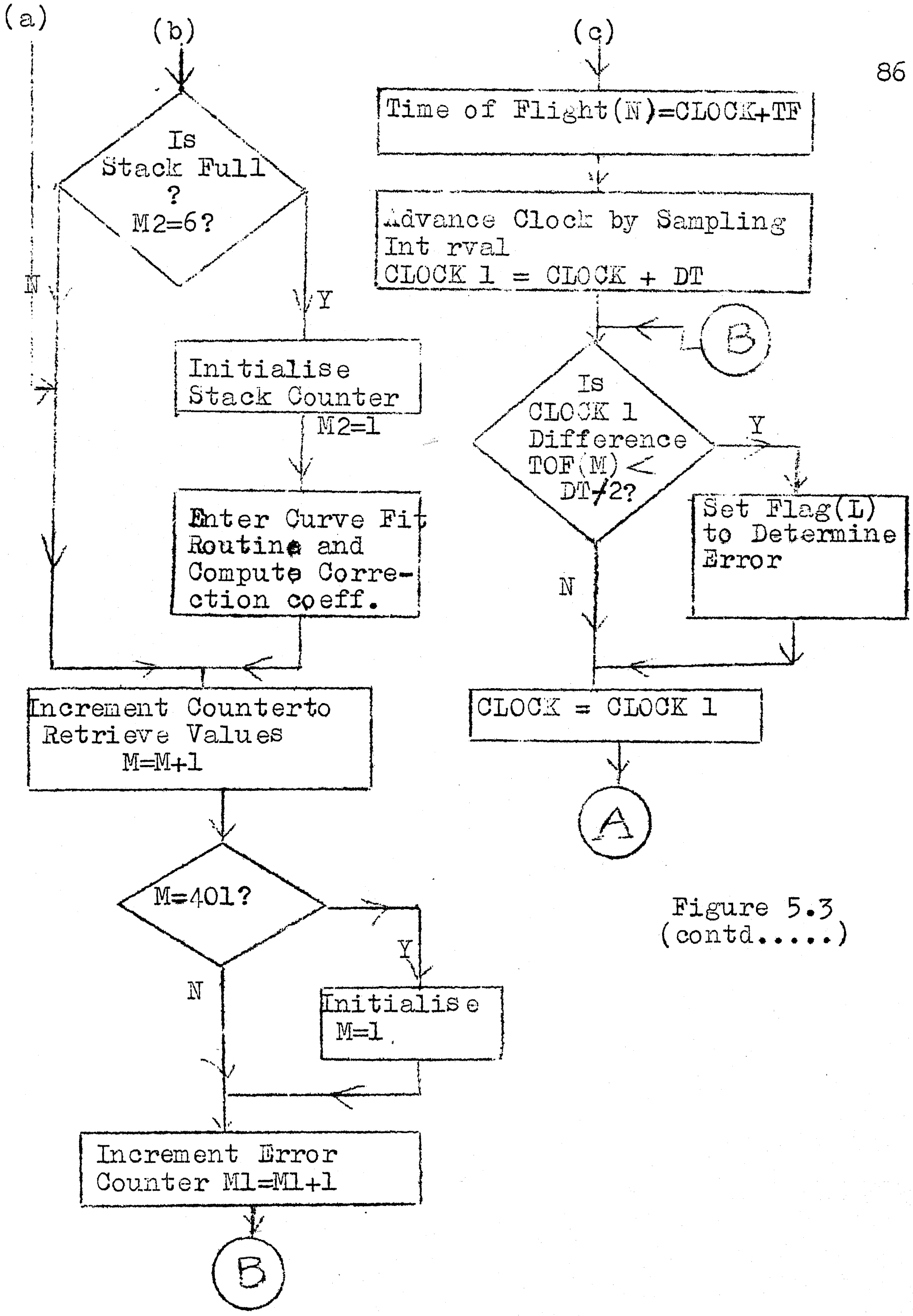


Figure 5.3
(contd.....)

the sampling interval. This has been found to work very satisfactorily for all the cases studied.

To get early values of the correction coefficients, the first six values of error computed are used for the curve fit routine and hence the correction coefficients. Subsequently, it is not necessary to enter the curve fit routine for each subsequent error value as this is redumdant [3] A procedure adopted is to enter each 7th value into stack and when the stack is full of 6 fresh values, enter curve fit routine and obtain fresh correction coefficients.

Any firing on the target causes an interrupt, and the program execution starts again at 'LOOP'.

6. CONCLUSION AND SCOPE FOR FURTHER WORK

The error correction technique employing first order polynomial curve fitting routine has been found to greatly enhance the accuracy of the system. Error in range in some cases is reduced to as low a value as 0.5 feet while that in bearing and elevation is reduced to almost negligible values. As the error function is continuously computed by fitting curves repetitively (when the stack of error values is full), fresh correction coefficients are continuously available and thus the error values are checked from continuously increasing.

Results also show that a microprocessor can easily be adapted to the solution of a real time application. A dedicated microprocessor, supported by a small amount of specialized interface hardware and a well designed software support is shown to offer a viable alternative to special purpose discrete logic system. The low cost of microprocessors provide a significant economic advantage to systems of this type. 8-bit microprocessor (INTEL 8080), with additional hardware, has been found suitable for floating point arithmetic employing a 24 bits mantissa and 6 bits exponent.

Further work suggested on this topic relates to

- o Simulation of the proposed design on one of the general purpose computers and hardware design of the system
- o Implementation of the system on one of the existing Fire Control Systems of the Navy.
- o Collecting sufficient data regarding shell trajectories through a number of experiments and trials and compiling tables which relate gun elevation and range to the time of flight. With these tables available studying the space -time trade off in calculating the time of flight by the iterative method used in our design and that by using Table look-up and interpolation.
- o With the present availability of a wide variety of 16 bits microprocessors, investigating the implementation of this design using a 16 bit microprocessor and eliminating the need of support hardware for floating point arithmetic.

REFERENCES

1. Naval Gunnery Manual, B.R. 1898, Part I, Volume 9, 'Predi tion System', Briti:h Admirality Publication [Restricted]

2. Naval Gunnery Manual, B.R. 1898, Part I, Volume 10, 'Ballistic Corrections', British Admirality Publication [Restricted].

3. BOOPATHI, A.M., 'Digital Computer Applications in a Weapon Control System', M.Tech. Thesis (EE), August, 1972

4. WRIGLEY, W. and J. HOVORKA, 'Fire Control Principles', McGraw Hill, 1959.

5. 'Intel 8080 Microcomputer Systems User's Manual', Sept. 1975.

6. SCHOEFFLER, J.D., 'Microprocessor Architecture', IEEE Transactions on Industrial Electronics and Control Instrumentation, Vol. IECI-22, No.3, Aug. 1975, pp. 256-272.

7. Lecture Notes on Intensive Course in Microprocessors', conducted by Dr. R.D. Solomon and Dr. S.K. Burns.

8. HILBURN, J.L. and R.M. JULICH, 'MICROCOMPUTERS/MICRO-PROCESSORS Hardware, Software and Applications', Prentice Hall Inc., N.J. 1976.

9. 'Intel 8080 Assembly Language Programming Manual'- Intel Corpn. Publication.

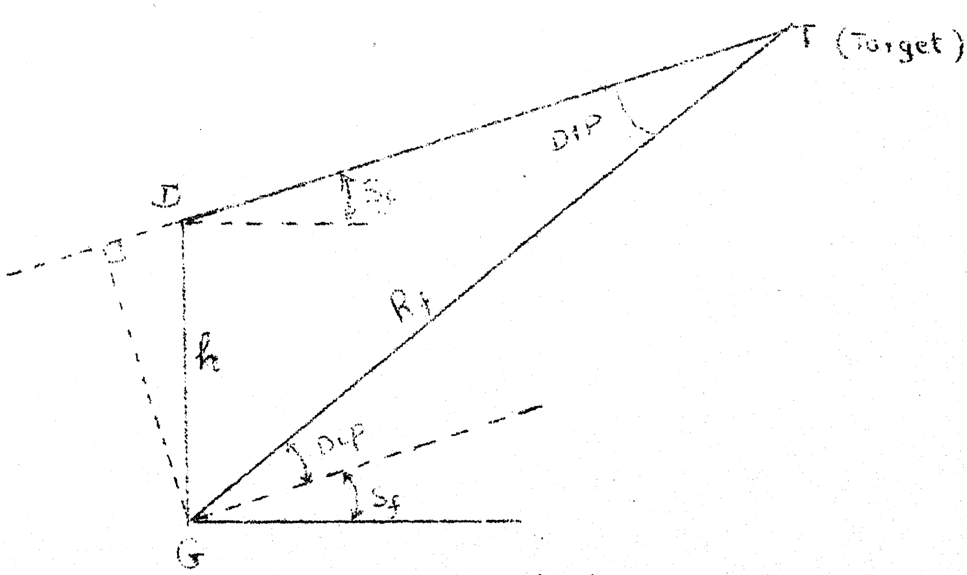
10. CARNAHAN, B., H.A. LUTHER and J.O. WILKES, 'Applied Numerical Methods', New York, Wiley, 1969.

APPENDIX A

Seperation Corrections

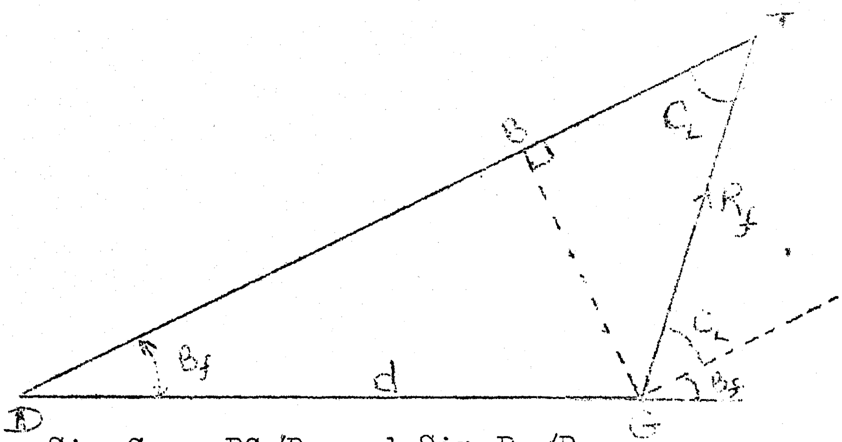
In the text it was assumed that the gun and the radar antenna are theoretically at the same point and plane. However, this is not so practically. To account for this separation of the gun (G) and director (D) the undermentioned corrections are applied.

(a) <u>DIP Correction :</u> The correction to elevation necessary because of the difference in height above deck-plane of director and gun. It can be defined as the angle between the director line of sight and the gun line of sight due to the vertical displacement (h) between the gun and the director.



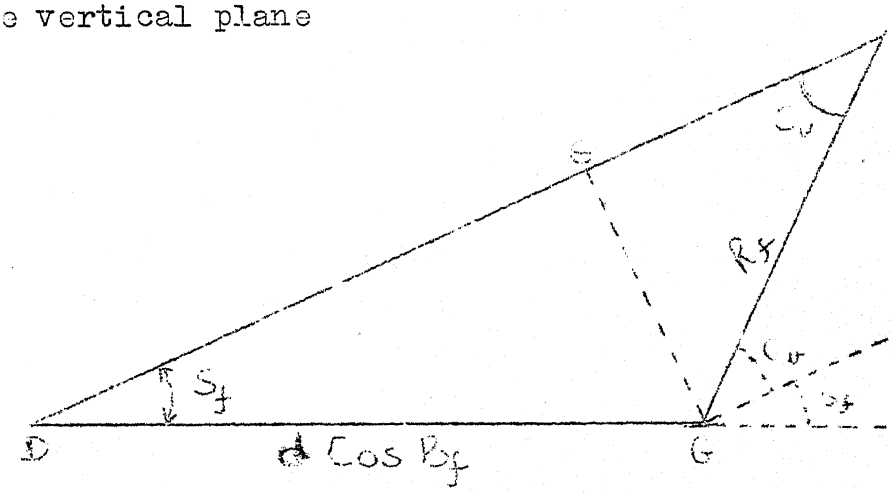
$$\text{Sin Dip} = SG/R_f = h \text{ Cos } (S_f)/R_f$$

(b) <u>Converrence Cirrection :</u> In A.A. fire this is required in both the horizontal and vertical planes. This is due to the horizontal displacement (d) between the director and the gun. Diagrammatically these are explained below :



$$\text{Sin } C_L = BG/R_f = d \text{ Sin } B_f/R_f$$

In the vertical plane



$$\text{Sin } C_v = SG/R_f = d \text{ Cos } B_f \text{ Sin } S_f/R_f$$

PREDICTION ERROR PROGRAM

```
------------------------------------------------------------------------
------------------------------------------------------------------------

PROGRAM TO COMPUTE ERRORS IN PREDICTION

------------------------------------------------------------------------
------------------------------------------------------------------------
      DIMENSION TOF(200),PRF(200),PBF(200),PSF(200),PRFC(200),PBFC(200),
     1PSFC(200)
      COMMON RP ,SP,BP,RRATE,SRATE,BRATE,DT,TF,V1,SF,BF,RRF,DTR,L,M
      COMMON G,JSTART,WR,WL
      COMMON AA(3),BB(3),STAC(6,3),RANGE(6),RFA(200),SFA(200),BFA(200)
      INTEGER P
      DTR = 22./1260.
      V1 = 2350.
      DT = 0.2
      G = 32.
      WRITE(6,100)
  100 FORMAT(//*------------------------------------------------------------
     1---------------------------------------------------------------*,
     2//*DIMENSION  FOR  PARAMETERS  ARE*,/*RANGE IN FEET*,9X,*RANGE RAT
     3E IN FEET PER SECOND*,/*ELEVATION IN DEGREES*,2X,*ELEVATION RATE I
     4N DEGREES PER SECOND*,/*BEARING IN DEGREES*,4X,*BEARING RATE IN DE
     5GREES PER SECOND*)
      NUMW = 0
      WRITE (6,101)
  101 FORMAT(/*TIME IN SECONDS*,/*WIND VELOCITY(W) IN FEET PER SECOND*,
     1/*WIND INCLINATION IN DEGREES(WI)*,/*RANGE ERROR IN FEET*,/*ELEVAT
     3ION AND BEARING ERRORS IN MINUTES*,//*----------------------------
     4------------------------------------------------------------------
     5-------------------------*///)
      DO 1111 I = 1,14
      ------------------------------------------------------------------

 RP, SP, BP, ARE THE PRESENT RANGE , ELEVATION , AND BEARING RESPECTIVEL
  RRATE,SRATE,BRATE,ARE THE COORESPONDING RATES
  DT IS THE SAMPLING INTERVAL
    WIND VELOCITY IS 0 TO 100 FEET PER SECOND
    WIND INCLINATION IS ASSUMED 10 DEGREES TO 70 DEGREES

      ------------------------------------------------------------------
      READ(5,10) RP,SP,BP,RRATE,SRATE,BRATE
   10 FORMAT(F10.2,F7.2,F8.2,F8.2,F7.2,F7.2)
      RP = RP*3.
      NUMW = NUMW + 1
      W = 100. *RNDY1(DUM1)
      WID = 10. + 60. * RNDY2(DUM2)
      WRITE(6,110) NUMW
  110 FORMAT(//* INPUT DATA *,I2,*.*)
      WRITE(6,103)W,WID,RRATE,SRATE,BRATE,RP,SP,BP
  103 FORMAT(*CONSTANT PARAMETERS ARE*,3X,* W = *,F12.5,3X,*WI = *,F12.5
     1,3X,*RRATE = *,F8.3,3X,*SRATE = *,F6.3,3X,*BRATE =*,F7.3,/*INITIAL
```

```
  2  PARAMETERS ARE*,3X,*RP = *,F12.3,3X,*SP = *,F7.3,8X,*   BP = *,
  3F7.3)
   WI = WID * DTR
   WR = W * COS ( WI )
   WL = W * SIN ( WI )
   ISTART = 1
   P = 0
   JSTART = 0
   ITRIG = 0
   N = 0
   CLOCK = 0
   M = 1
   L = 0
   ICOUNT = 0
   LCOUNT = 0
   K = 0
   M1 = 1
   M2 = 1
   FIREAT = 1000.
   RELAX = 2. * RP
   IF( RP .LT. 20001. ) RELAX = 3. * RP
   SP = SP * DTR
   BP = BP * DTR
   BRATE = BRATE*DTR
   SRATE = SRATE*DTR
------------------------------------------------------------------------

 L IS A FLAG TO DETERMINE IF THE CLOCK VALUE CORRESPONDS TO A VALUE OF
 COMPUTED TIME OF FLIGHT

------------------------------------------------------------------------
 20 IF ( L .EQ. 1 ) GO TO 40
    N = N+ 1
    CALL TIME
    IF(JSTART .EQ. 1 ) GO TO 11
    JSTART = 1
    WRITE (6,104) TF
104 FORMAT(1X,*INITIAL TIME OF FLIGHT = *,F10.4)
 11 IF ( RP .LE. FIREAT ) ITRIG = 1
    IF ( ITRIG .EQ. 1 ) GO TO 1110
    IF ( RRF .EQ. 0. ) GO TO 1111
     IF ( RP .GE. RELAX ) GO TO 1110
    PRF(N)= RRF
    PSF(N) = SF
    PBF(N) = BF
    RFA(N) = RP + RRATE * TF
    DV = ARSIN ( RP / RFA(N) * SRATE * TF )
    SFA(N) = SP + DV
    DA = ARSIN(RP*COS(SP)*BRATE*TF/(RFA(N)*COS(SFA(N))))
    BFA(N) = BP + DA
    IF(BFA(N) .GT. 6.28 )  BFA(N) = BFA(N) - 6.28
    IF(BFA(N) .LT. 0. )  BFA(N) = 6.28 + BFA(N)
    IF(K .EQ. 1 ) GO TO 22
    PRFC(N) = PRF(N)
```

```
      PSFC(N) = PSF(N)
      PBFC(N) = PBF(N)
      GO TO 21
   22 CALL CORECT (PRF(N),DRX,DSX,DBX)
      PRFC(N) = PRF(N) - DRX
      PSFC(N) = PSF(N) - DSX
      PBFC(N) = PBF(N) - DBX
   21 TOF(N) = CLOCK + TF
      CLOCK1 = CLOCK + DT
   30 DIFF = ABS(CLOCK1 - TOF(M) )
      IF ( DIFF .LT. (DT/2.) ) GO TO 31
      CLOCK = CLOCK1
      GO TO 32
-------------------------------------------------------------------------

 IF L = 1 THEN THE VALUES OBTAINED FROM INPUT(RADAR) ARE COMPARED WITH
 THE COMPUTED VALUES AT THAT INSTANCE TO DBTAIN THE RELEVANT ERRORS.

-------------------------------------------------------------------------
   31 L = 1
      IF ( ISTART .EQ. 0 ) GO TO 20
      ISTART = 0
   32 CALL COMPUT(CRF,CSF,CBF)
-------------------------------------------------------------------------

  COMPUT RETURNS THE FUTURE POSITION COORDINATES AFTER SAMPLING INTERVAL
 TIME ADVANCED BY SAMPLING INTERVAL DT
 THE PRESENT COORDINATES NOW BECOME THE COORDINATES AFTER SAMPLING TIME

-------------------------------------------------------------------------
      RP = CRF
      BP = CBF
      SP = CSF
      IF ( N .EQ.  200 ) N = 0
      GO TO 20
   40 CALL ERRORS(RE,EE,BE,PRF(M),PSF(M),PBF(M))
      DR = RE
      DE = EE
      DB = BE
      DEM = DE / DTR * 60.
      DBM = DB / DTR * 60.
      IF ( K .EQ. 0 ) GO TO 41
-------------------------------------------------------------------------

      WHEN K = 1 CURFIT ROUTINE HAS BEEN ENTERED HENCE CORRECTED PARAMETER

-------------------------------------------------------------------------
      CALL ERRORS(RE,EE,BE,PRFC(M),PSFC(M),PBFC(M))
      DRC = RE
      DEC = EE
      DBC = BE
      DECM = DEC / DTR * 60.
      DBCM = DBC / DTR * 60.
      IF ( DR .EQ. DRC ) GO TO 41
```

```
      LCOUNT = LCOUNT + 1
      IF(LCOUNT .EQ. 10 ) LCOUNT = 1
      IF(LCOUNT .NE. 1 ) GO TO 41
      WRITE(6,105)CLOCK1,DR,DRC,DEM,DECM,DBM,DBCM
  105 FORMAT(/*AFTER *,F10.4,* SECONDS OF DETECTION *,
     1/* UNCORRECTED RANGE ERROR     = *,F11.5,10X,*CORRECTED = *,F11.5,
     2/* UNCORRECTED ELEVATION ERROR = *,F11.5,10X,*CORRECTED = *,F11.5,
     3/* UNCORRECTED BEARING ERROR   = *,F11.5,10X,*CORRECTED = *,F11.5)
      ICOUNT = ICOUNT + 1
      IF( ICOUNT .EQ. 6 ) GO TO 1110
   41 L = 0
      IF ( M1 .EQ. 6 ) GO TO 42
      IF ( P .EQ. 0 ) GO TO 43
      GO TO 44
---------------------------------------------------------------------

      P = 0 FOR M .LE. 9 AND THEN IS ALWAYS 1

---------------------------------------------------------------------
   42 P = 1
      M1 = 1
---------------------------------------------------------------------

      M1 IS A COUNTER THAT DECIDES ENTRY INTO DTACK

---------------------------------------------------------------------
   43 RANGE(M2) = PRF(M)
      STAC(M2,1) = DR
      STAC(M2,2) = DE
      STAC(M2,3) = DB
      M2 = M2 + 1
---------------------------------------------------------------------

      M2 KEEPS TRACK OF NUMBER OF ENTRIES IN STACK

---------------------------------------------------------------------
      IF ( M2 .EQ. 7 ) GO TO 45
   44 M = M+ 1
      IF ( M .EQ. 201 ) M = 1
      M1 = M1 + 1
      GO TO 30
   45 M2 = 1
---------------------------------------------------------------------

      STACK POINTER INITIALISED

---------------------------------------------------------------------
      CALL CURFIT
      K = 1
      GO TO 44
  110 WRITE(6,106)CLOCK1,TF
  106 FORMAT(/*FIRED AT TARGET AFTER *,F10.4,* SECONDS OF DETECTION.
     1TIME OF FLIGHT = *,F10.4)
  111 CONTINUE
      STOP
      END
```

```
      SUBROUTINE TIME
      COMMON RP ,SP,BP,RRATE,SRATE,BRATE,DT,TF,V1,SF,BF,RRF,DTR,L,M
      COMMON G,JSTART,WR,WL
      COMMON AA(3),BB(3),STAC(6,3),RANGE(6),RFA(200),SFA(200),BFA(200)
      REAL NEWT,NEWR
      N = 1
      IF ( JSTART .EQ. 1 ) GO TO 20
      V = V1
      OLDR = RP
      OLDT = RP / V
      SF = SP
   20 A = G * SIN(SF)
      IF(OLDR  .LT. 45000.)  V= V1
      IF(OLDR   .GT. 60000.)  V = 10. * V1
      B = -2. * V
      C = 2. * OLDR
      X1 = 0.
      X2 = 0.
      CALL ROOTS (A,B,C,X1,X2)
      IF((X1 .GT. 0.) .AND. (X2 .LT. 0.))  GO TO 30
      IF((X1 .GT. 0.) .AND. (X2 .GT. 0.))  GO TO 32
      GO TO 444
   30  NEWT = X1
      GO TO 31
   32  NEWT = X2
   31 TERM1 = RRATE * NEWT
      NEWR = RP + TERM1
      IF(ABS(NEWT  -  OLDT) .GT. 0.05) GO TO 40
      IF(ABS(NEWR  -  OLDR) .GT. 1.00) GO TO 40
      GO TO 50
   40 N = N+1
      IF ( N .EQ. 15) GO TO 555
      OLDT = NEWT
      OLDR = NEWR
      EXP1 = RP/NEWR * SRATE *NEWT
      IF (EXP1 .GT. 1. )  EXP1 = 0.98
      IF (EXP1 .LT. (-1.) ) EXP1 = (-.98 )
      DV = ARSIN(EXP1)
      DVD = DV/DTR
      SF = SP + DV
      IF((NEWR  .LT. 30000.) .AND. (SF .LT. .174603)) GO TO 42
      IF((NEWR  .LT. 30000.) .AND. (SF .GT. 1.396824)) GO TO 43
      IF (SF .LT. 0. ) GO TO 44
      IF(SF .GT. 1.571427) SF = 1.54
   41 SFD = SF/DTR
      GO TO 20
   42  SF = .174603
       GO TO 41
   43  SF = 1.3619034
      GO TO 41
   44 IF (NEWR .LT. 60000. ) SF = 0.0174603
      SF = SF - DV
      GO TO 41
  444 SF = SF + 5.*DTR
```

```
      IF (SF .GT. 1.57 ) GO TO 555
      GO TO 41
555   WRITE(6,556)
556   FORMAT(1H0,10X,*LOST  TARGET *)
      RRF = 0.
      RETURN
 50   EXP1 =      (RP*COS(SP)*BRATE*NEWT  )/(NEWR *COS(SF))
      IF (EXP1 .GT. 1. )  EXP1 = 0.98
      IF (EXP1 .LT. (-1.) ) EXP1 = (-.98 )
      DA = ARSIN(EXP1)
      BF = BP + DA
      BFD = BF/DTR
      IF (BFD .LT. 0. ) BFD = 360. + BFD
      IF ( BFD .GT. 360. ) BFD = BFD - 360.
      BF = BFD * DTR
      TF = NEWT
      RRF = NEWR
------------------------------------------------------------------------

      INTRODUCING CORRECTIONS

------------------------------------------------------------------------
      TANELE = G * TF * COS(SF) / ( 2. * V)
      PHI = SF + TANELE
      DRICON = 0.04
      DRIFT = DRICON * TANELE
      BF = BF - DRIFT
      DELV = 0.10
      DELC = 0.10
------------------------------------------------------------------------

      CORRECTIONS DUE TO CHANGES IN MUZZLE VELOCITY AND BALLISTICS

------------------------------------------------------------------------
      IF (JSTART .EQ. 1 )  GO TO 203
      DELTM = RNDY1(DUM1) * DELV + RNDY2(DUM2) * DELC
      DELDVM = (RNDY2(DUM2)*DELV + RNDY1(DUM1) * DELC ) * 0.1
------------------------------------------------------------------------

      EFFECTS DUE TO WIND
      CHANGE IN RANGE,ELEVATION AND BEARING DUE TO WIND

------------------------------------------------------------------------
203   DELRW = WR * TF * COS ( PHI )
      DELPHI = - ( WR * TF * SIN( PHI ) ) / RRF
      DELAW = ( WL * TF ) / ( RRF * COS ( SF ) )
------------------------------------------------------------------------

      THEREFORE FINAL FUTURE RANGE ELEVATION AND BEARING ARE

------------------------------------------------------------------------
204   RRF = RRF + DELRW
      GUNELE = PHI + DELPHI
      BF = BF + DELAW
```

```
      IF(BF .GT. 6.2857) BF = BF - 6.2857
      IF(BF .LT. 0. )  BF = BF + 6.2857
C ----------------------------------------------------------------------
C
C     THE RANGE CORRECTION IS EFFECTED BY CORRECTION TO TIME OF FLIGHT
C     AND ELEVATION AS FOLLOWS
C
C ----------------------------------------------------------------------
      DELTW = WR * TF * COS(SF) / V
      DELDVW = -((WR * COS(SF) ) / ( V * SIN(PHI) * COS(PHI) ) )
C ----------------------------------------------------------------------
C
C     THEREFORE FINAL TF AND SF ARE
C
C ----------------------------------------------------------------------
  205 TF = TF + DELTW + DELTM
      SF = SF + DELDVW + DELDVM
      RETURN
      END
      SUBROUTINE ROOTS (A,B,C,X1,X2)
      IF(A .EQ. 0.)  RETURN
      IF(B*B .LT. 4.*A*C) RETURN
      X1 = (-B +SQRT(B*B-(4.*A*C)))/(2.*A)
      X2 = (-B -SQRT(B*B-(4.*A*C)))/(2.*A)
      RETURN
      END
```

AIRCRAFT SIMULATION ROUTINE

```
      SUBROUTINE COMPUT(CRF,CSF,CBF)
      COMMON RP ,SP,BP,RRATE,SRATE,BRATE,DT,TF,V1,SF,BF,RRF,DTR,L,M
      COMMON G,JSTART,WR,WL
      COMMON AA(3),BB(3),STAC(6,3),RANGE(6),RFA(200),SFA(200),BFA(200)
      TERM1 = RRATE * DT
      TERM2 = 0.5 * RP*DT *DT
      TERM3=SRATE*SRATE+BRATE*BRATE*COS(SP)*COS(SP)
      TERM3 = 0.
      TERM2 = TERM2 * TERM3
      CRF = RP + TERM1 + TERM2
      DV = ARSIN (RP/CRF*SRATE*DT)
      CSF = SP + DV
      DA = ARSIN( RP*COS(SP)*BRATE*DT/(CRF*COS(CSF)))
      CBF = BP + DA
      CBFD = CBF/DTR
      IF (CBFD .LT. 0. ) CBFD = 360. + CBFD
      IF (CBFD .GT. 360. ) CBFD = CBFD - 360.
      CBF = CBFD * DTR
      RETURN
      END
```

ERRORS ROUTINE

```
      SUBROUTINE ERRORS (RE,EE,BE,PRF,PSF,PBF)
      COMMON RP ,SP,BP,RRATE,SRATE,BRATE,DT,TF,V1,SF,BF,RRF,DTR,L,M
      COMMON G,JSTART,WR,WL
      COMMON AA(3),BB(3),STAC(6,3),RANGE(6),RFA(200),SFA(200),BFA(200)
      RE = PRF - RFA(M)
      EE = PSF - SFA(M)
      BE = PBF - BFA(M)
      IF (BE .GT. 3.14 ) BE = 6.28 - BE
      IF ( BE .LT. ( -3.14 ) )  BE = 6.28 + BE
      RETURN
      END
$IBFTC SUB5
```

CURVE FITTING ROUTINE

```
      SUBROUTINE CURFIT
      COMMON RP ,SP,BP,RRATE,SRATE,BRATE,DT,TF,V1,SF,BF,RRF,DTR,L,M
      COMMON G,JSTART,WR,WL
      COMMON AA(3),BB(3),STAC(6,3),RANGE(6),RFA(200),SFA(200),BFA(200)
      DIMENSION A0(3),A1(3)
      L = 1
   10 I = 1
      J = 1
       DO 1 K = 1,3
      X1 = RANGE(I)
      X2 = RANGE(I+1)
      Y1 = STAC(I,L)
      Y2 = STAC(I+1,L)
      D = X2 - X1
      A0(J) = (X2 * Y1 - X1 * Y2 ) / D
      A1(J) = ( -Y1 + Y2 ) / D
      J = J + 1
      I = I + 2
    1 CONTINUE
      AA(L) = (A0(1) + A0(2) + A0(3) ) / 3.
      BB(L) = (A1(1) + A1(2) + A1(3) ) / 3.
      L = L + 1
      IF ( L .LE. 3 ) GO TO 10
      RETURN
      END
```

APPENDIX C

```
;*****************************************************************

;MACROS USED IN THE ASSEMBLY LANGUAGE PROGRAM OF THE SYSTEM

;*****************************************************************

;MACRO NAME IS IN THE LABEL FIELD

TRANS   MACRO   LOC1,LOC2               ;MACRO TO TRANSFER 4 BYTES
                                        ;FROM MEMORY LOCATION LOC1
                                        ;TO MEMORY LOCATION LOC2
        LHLD    LOC1
        SHLD    LOC2
        LHLD    LOC1 + 2
        SHLD    LOC2 + 2
        ENDM


INSRT   MACRO   SPX                     ;MACRO TO INSERT A 4 BYTE
                                        ;WORD INTO STACK
                                        ;SPX INDICATES CURRENT VAL
                                        ;OF STACK POINTER
        MOV     B,H
        MOV     C,L
        LHLD    SPX
        SPHL
        PUSH    D
        PUSH    B
        DCX     H                       ;STACK POINTER ADJUSTED
        DCX     H                       ;TO INSERT NEXT VALUE
        DCX     H
        DCX     H
        SHLD    SPX
        ENDM


RETRV   MACRO   IPX                     ;MACRO TO RETRIEVE A 4
                                        ;BYTE WORD FROM STACK
                                        ;IPX IS POINTER TO BYTE
                                        ;TO BE REMOVED
        LHLD    IPX
        SPHL
        POP     H
        POP     D
        ENDM


DETER   MACRO   ADDR1,ADDR2             ;MACRO TO DETERMINE THE
                                        ;DETERMINANT TO FIND A
```

```
                                                  ;SOLUTION FOR OBTAINING
                                                  ;COEFFICIENTS OF FIRST
                                                  ;ORDER POLYNOMIAL
                                                  ;ADDR1,ADDR2 REFER TO TWO
                                                  ;RANGE VALUES
        LXI     SP,ADDR1
        POP     H
        POP     D
        SHLD    X1
        XCHG
        SHLD    X1 + 2
        LXI     SP,ADDR2
        POP     H
        POP     D
        SHLD    X2
        XCHG
        SHLD    X2 + 2
        XCHG
        FSUB    X1,D
        SHLD    DET
        XCHG
        SHLD    DET + 2
        ENDM


COEFF   MACRO   ADDR1,ADDR2,ST01,ST02             ;MACRO TO COMPUTE THE TWO
                                                  ;COEFFICIENTS OF A FIRST
                                                  ;ORDER PILYNOMIAL
                                                  ;X1,X2 ARE TWO RANGE VALUE
                                                  ;STACK HAS ERROR VALUES
                                                  ;COEFF. STORED IN ST01,ST02
                                                  ;ADDR1,ADDR2 ARE POINTERS
                                                  ;TO ERROR VALUES
        LXI     SP,ADDR1
        POP     H
        POP     D
        FMUL    X2,D
        SHLD    TEMP
        XCHG
        SHLD    TEMP + 2
        LXI     SP,ADDR2
        POP     H
        POP     D
        FMUL    X1,D
        FSUB    D,TEMP
        FDIV    DET,D
        SHLD    ST01
        XCHG
        SHLD    ST01 + 2
        LXI     SP,ADDR1
        POP     H
        POP     D
        FSUB    D,ADDR2
        FDIV    DET,D
        SHLD    ST02
```

```
        XCHG
        SHLD    ST02 + 2
        ENDM


AVRGE   MACRO   LOC1,LOC2,LOC3,LOC4          ;MACRO TO DETERMINE THE
                                             ;FINAL COEFF,BY AVERAGING
                                             ;3 SETS OF COMPUTED COEFF.
                                             ;LOC1,LOC2,LOC3 CONTAIN
                                             ;THE VALUES TO BE AVERAGED
                                             ;LOC4 STORES FINAL COEFF.
        LHLD    LOC1 + 2
        XCHG
        LHLD    LOC1
        FADD    LOC2,D
        FADD    LOC3,D
        FDIV        ,D                       ;DIVIDE BY 3
        SHLD    LOC4
        XCHG
        SHLD    LOC4 + 2
        ENDM


DECRT   MACRO   IPT                          ;MACRO TO DECREMENT THE
                                             ;VALUE OF A POINTER BY 4
        LHLD    IPT
        DCX     H
        DCX     H
        DCX     H
        DCX     H
        SHLD    IPT
        ENDM


RTVAL   MACRO   ARG1,ARG2                    ;MACRO TO STORE THE RIGHT
                                             ;VALUES OF PREDICTED COORD
                                             ;ARG1 CORRESPONDS TO THE
                                             ;COORDINATE TO BE CORRECTED
                                             ;AND ARG2 TO THE CORRECTIO
                                             ;FACTOR.
        LHLD    ARG1 +2
        XCHG
        LHLD    ARG1
        FSUB    ARG2,D
        SHLD    ARG1
        XCHG
        SHLD    ARG1 + 2
        ENDM
```

```
                                        ;THE CORRECTION COEFF(STO)
                                        COEF1,COEF2 ARE THE COEFFS
                                        ;OF THE EQUATION
                                        ;ERROR=COEF1+RANGE*COEF2
        LHLD    NEWR + 2
        XCHG
        LHLD    NEWR
        FMUL    D,COEF2
        FADD    COEF1
        SHLD    STO
        XCHG
        SHLD    STO + 2
        ENDM
```

```
;*************************************************************

;MAIN PROGRAM IN ASSEMBLY LANGUAGE OF 8080

;*************************************************************

;PROGRAM ASSUMES RP,SP,BP,R,S,B,ARE CONTINUOUSLY AVAILABLE
;IN MEMORY LOCATIONS SPECIFIED FOR THESE.

;SETTINGS FOR W,WI,V1,DELV,CELC ARE TRANSFERRED TO MEMORY
;LOCATIONS RESERVED FOR THESE

;*************************************************************

;INITIALISATION

;*************************************************************

START:  XRA     A                           ; VALUES INITIALISED
        STA     N                           ; TO ZERO
        STA     JSTAR
        STA     P
        STA     K
        STA     L
        MOV     H,A
        MOV     L,A
        SHLD    CLOCK
        SHLD    CLOCK + 2
        MVI     A,1                         ; VALUES INITIALISED
        STA     M                           ; TO ONE
        STA     M1
        STA     M2
        LXI     H,2761H                     ;POINTERS INITIALISED
        SHLD    SP1                         ;STORAGE POINTER FOR PRF
        LXI     H,2121H
        SHLD    SP2                         ;STORAGE POINTER FOR PSF
        LXI     H,1AE1H
        SHLD    SP3                         ;STORAGE POINTER FOR PBF
        LXI     H,14A1H
        SHLD    SP4                         ;STORAGE POINTER FOR TOF
        LXI     H,275DH
        SHLD    IPT1                        ;RETREIVE POINTER FOR PRF
        LXI     H,211DH
        SHLD    IPT2                        ;RETREIVE POINTER FOR PSF
        LXI     H,1ADDH
        SHLD    IPT3                        ;RETREIVE POINTER FOR PBF
        LXI     H,149DH
```

```
        SHLD    IPT4                    ;RETREIVE POINTER FOR TOF
        LXI     H,0E61H
        SHLD    STAC1                   ;POINTER FOR DR
        LXI     H,0E49H
        SHLD    STAC2                   ;POINTER FOR DE
        LXI     H,0E31H
        SHLD    STAC3                   ;POINTER FOR DB
        LXI     H,0E19H
        SHLD    STAC4                   ;POINTER FOR RANGE

      ; COMPUTE WIND COMPONENTS

      ;EQUIVALENT WIND VELOCITY(W) IN LOCATION 0D1DH

      ; WIND INCLINATION(WI) IN LOCATION 0D19H

        LHLD    0D1BH
        XCHG
        LHLD    0D19H
        COS
        FMUL    D,0D1DH
        SHLD    WR                      ; WR = W * COS(WI)
        XCHG
        SHLD    WR + 2
        LHLD    0D1BH
        XCHG
        LHLD    0D19H
        SIN
        FMUL    D,0D1DH
        SHLD    WL                      ; WL = W * SIN(WI)
        XCHG
        SHLD    WL + 2

      ; CHECK IF ERROR FLAG IS SET

LOOP:   LDA     L
        JNZ     ERORS                   ; FLAG SET IF NOT ZERO
        LDA     N
        INR     A
        MVI     B,401
        CMP     B
        JNZ     CONTN                   ; CONTINUE IF N .LE. 400

      ; N .GT. 400,THEREFORE INITIALISE ALLSTORE POINTERS AND N

        LXI     H,2761H
        SHLD    SP1
        LXI     H,2121H
        SHLD    SP2
        LXI     H,1AE1H
        SHLD    SP3
        LXI     H,14A1H
        SHLD    SP4
        MVI     A,1
```

```
       ;TIME RETURNS THE TIME OF FLIGHT AND PREDICTED COORDINATES

        MVI      A,1
        STA      JSTAR
TIMER:  LHLD     NEWR + 2                     ;CHECK NEWR
        MOV      A,H                          ; IF NEWR = 0 TARGET IS LOS
        JNZ      KCHEK
        XCHG
        LHLD     NEWR
        MOV      A,H
        JNZ      KCHEK
        JMP      START
KCHEK:  LHLD     NEWR+ 2                      ; PUT PREDICTED COORDDS.
        XCHG                                  ; IN THEIR RESPECTIVE
        LHLD     NEWR                         ; STACKS
        INSRT    SP1
        LHLD     SF + 2
        XCHG
        LHLD     SF
        INSRT    SP2
        LHLD     BF + 2
        XCHG
        LHLD     BF
        INSRT    SP3
        LDA      K
        JZ       PRDCT
        JMP      CORCT                        ;STORE CORRECTED PREDICTED
                                              ;VALUES

       ; CORCT RETURNS THE CORRECTION FACTOR

CORCR:  RTVAL    NEWR,DRX
        RTVAL    SF,DSX
        RTVAL    BF,DBX
PRDCT:  LHLD     CLOCK + 2                    ;INCREMENT BY VALUE OF CL
        XCHG
        LHLD     CLOCK
        FADD     D,NEWT
        INSRT    SP4
        LHLD     CLOCK
        FADD     D,0D99H
        SHLD     CLCK1
        XCHG                                  ; CLCKZ = CLOCK + DT
        SHLD     CLCK1 + 2

       ; CLOCK1 AND TOF(M) COMPARED

TLOOP:  RETRV    IPT4
        FSUB     D,CLCK1
        MOV      A,D
        JP       ABSOL
```

```
        TRANS   CLCK1,CLOCK
        JMP     ILOOP
LFLAG:  MVI     A,1
        STA     L
        JMP     ILOOP

;*************************************************************

; ERROR CALCULATION ROUTINE

;*************************************************************

ERORS:  XRA     A
        STA     L
        LDA     M1
        MVI     B,6                 LOAD FIRST 6 VALUES IN STA
        CMP     B                   ;AND THEN EVERY SIXTH VAL.
        JZ      RESTO
        LDA     P
        JZ      PILE
        JMP     INCRM
RESTO:  MVI     A,1
        STA     P
        STA     M1
PILE:   RETRV   IPT1                ;RANGE FOR WHICH ERROR IS
        INSRT   STAC4               ;COMPUTED IS STORED IN STA
        MOV     H,B
        MOV     L,C
        FSUB    0D01H,D             ;ERRORS COMPUTED AND STORE
        INSRT   STAC1               ; IN STACK
        RETRV   IPT2
        FSUB    0D05H,D
        INSRT   STAC2
        RETRV   IPT3
        FSUB    0D09H,D
        INSRT   STAC3
        LDA     A,M2
        INR     A
        MVI     B,7
        CMP     B
        JZ      STAIN               ;STACK FULL,INITIALISE
        STA     M2
        JMP     INCRM
STAIN:  MVI     A,1
        STA     M2
        STA     K                   ;CORRECTION FLAG IS SET
        LXI     H,0E61H             ;INITIALISE STACK POINTERS
        SHLD    STAC1
        LXI     H,0E49H
        SHLD    STAC2
```

```
        ;CURVE FITTING ROUTINE

        ;*************************************************************************
         DETER    0E15H,0E11H                         ;GET THE DETERMINANT
         COEFF    0E5DH,0E59H,A0R1,A1R1
         COEFF    0E45H,0E41H,A0E1,A1E1
         COEFF    0E2DH,0E29H,A0B1,A1B1
         DETER    0E0DH,0E09H
         COEFF    0E55H,0E51H,A0R2,A1R2
         COEFF    0E3DH,0E39H,A0E2,A1E2
         COEFF    0E25H,0E21H,A0B2,A1B2
         DETER    0E05H,0E01H
         COEFF    0E4DH,0E49H,A0R3,A1R3
         COEFF    0E35H,0E31H,A0E3,A1E3
         COEFF    0E1DH,0E19H,A0B3,A1B3
         AVRGE    A0R1,A0R2,A0R3,AAR
         AVRGE    A0E1,A0E2,A0E3,AAE
         AVRGE    A0B1,A0B2,A0B3,AAB
         AVRGE    A1R1,A1R2,A1R3,BBR
         AVRGE    A1E1,A1E2,A1E3,BBE
         AVRGE    A1B1,A1B2,A1B3,BBB

        ;END OF CURVE FIT ROUTINE,INCREMENT RETRIEVE POINTER M.

        ;DECREMENT RETREIVAL STACK POINTERS BY 4

INCRM:   LDA      M
         INR      A
         MVI      B,401
         CMP      B
         JZ       INITM
         STA      M
         LDA      M1
         INR      A
         STA      M1
         DECRT    IPT1
         DECRT    IPT2
         DECRT    IPT3
         DECRT    IPT4
         JMP      TLOOP
INITM:   MVI      A,1
         STA      M
         LXI      H,275DH
         SHLD     IPT1
         LXI      H,211DH
         SHLD     IPT2
         LXI      H,1ADDH
         SHLD     IPT3
         LXI      H,149DH
         SHLD     IPT4
         JMP      TLOOP

        ;*************************************************************************
```

```
;;CORRECT ROUTINE, THIS DETERMINES THE CORRECTION FACTORS,

;*****************************************************************

CORCT: FUNC    AAR,BBR,DRX
       FUNC    AAE,BBE,DSX
       FUNC    AAB,BBB,DBX
       JMP     CORCR


;*****************************************************************

; TIME  ROUTINE

;*****************************************************************

;THIS COMPUTES THE TIME OF FLIGHT AND THE FUTURE COORDINATES


TIME:  MVI     A,15
       STA     NN
       LDA     JSTAR
       JNZ     LOOP
       TRANS   0D25H,V                     ; V = V1
       TRANS   0D01H,OLDR
       TRANS   0D05H,SF
       LHLD    0D27H
       XCHG
       LHLD    0D25H
       FDIV    D,0D01H
       SHLD    OLDT
       XCHG
       SHLD    OLDT + 2
LOOP:  LHLD    SF + 2
       XCHG
       LHLD    SF
       SIN
       FMUL    D,0D55H
       SHLD    AA
       XCHG
       SHLD    AA + 2                      ; AA = G * SIN(SF)

;CHECK IF RANGE ,GT, 60000 FEET OR .LT, 45000 FEET
;TO FIX MUZZLE VELOCITY.

       LHLD    OLDR + 2
       XCHG
       LHLD    OLDR
       FSUB    D,0D91H
       JM      INCRV
       TRANS   0D25H,V                     ;V = V1 IF .LT. 45000 FEET
       JMP     CALCB
INCRV: LHLD    OLDR + 2
```

```
        XCHG
        LHLD    OLDR
        FSUB    D,0D95H                 ; 60000 - OLDR
        JP      CALCB
        LHLD    0D27H
        XCHG
        LHLD    0D25H
        FMUL    D,0D89H                 ; V1 * 10
        SHLD    V
        XCHG
        SHLD    V + 2                   ; V = 10*V1 IF OLDR .GT. 6
CALCB:  LHLD    V + 2
        XCHG
        LHLD    V
        FMUL    D,0D79H
        SHLD    BB                      ;BB = -2 * V
        XCHG
        SHLD    BB + 2
        LHLD    OLDR + 2
        XCHG
        LHLD    OLDR
        FMUL    D,0D75H
        SHLD    CC
        XCHG
        SHLD    CC + 2                  ; CC = 2 * OLDR
        JMP     ROOTS

;*************************************************************************

;ROOTS RETURNS VALUE OF TIME OF FLIGHT IN NEWT

;*************************************************************************


ROOTR:  LHLD    NEWT + 2
        XCHG
        LHLD    NEWT
        FSUB    D,0D6DH                 ; 0 - NEWT
        MOV     A,D
        JP      LOOP2                   ;IF NEWT = 0 ,TRY AGAIN
                                        ;ALTERING SF
        LHLD    NEWT + 2
        XCHG
        LHLD    NEWT
        FMUL    D,0D0DH
        FADD    D,0D01H
        SHLD    NEWR                    ;NEWR = RP + R * TF
        XCHG
        SHLD    NEWR + 2

;COMPARE TO SEE IF TWO SUCCESSIVE ITERATIONS ARE WITHIN LIMITS

        LHLD    NEWT + 2
        XCHG
        LHLD    NEWT
        FSUB    D,OLDT
```

```
        MOV     A,D
        JP      COMPT
        FSUB    D,0D6DH                       ; 0 - (OLDT - NEWT)
                                              ;TO GET ABSOLUTE ERROR
COMPT:  FSUB    D,0D45H                       ; 0.05 - DIFF
        JM      NEXT                          ; DIFF .GT. 0.05
        LHLD    NEWR + 2
        XCHG
        LHLD    NEWR
        FSUB    D,OLDR
        MOV     A,D
        JP      COMPR
        FSUB    D,0D6DH                       ; FOR ABSOLUTE ERROR
COMPR:  FSUB    D,0D71H
        JM      NEXT                          ;DIFF .GT. 1 FOOT

;DIFFERENCE IS BETWEEN LIMITS AND HENCE ITERATIONS ARE COMPLETE

;COMPUTE DA TO DETERMINE BF

;DA = ARSIN((RP*COS(SP)*B*TF)/(RF*COS(SF))

        LHLD    SF + 2
        XCHG
        LHLD    SF
        COS
        FMUL    D,NEWR
        SHLD    TEMP
        XCHG
        SHLD    TEMP + 2                      ;TEMP = NEWR * COS(SF)
        LHLD    0D07H
        XCHG
        LHLD    0D05H
        COS
        FMUL    D,0D01H
        FMUL    D,0D15H
        FMUL    D,NEWT
        FDIV    TEMP,D
        ARSIN
        FADD    D,0D09H                       ;BF = BP + DA

;LIMIT BF TO BE WITHIN 0 TO 360 DEGREES

        MOV     A,D
        JM      BFNEG
        FSUB    0D4DH,D                       ;BF - 6.28
        JP      STORB
BFNEG:  FADD    0D4DH,D
STORB:  SHLD    BF
        XCHG
        SHLD    BF + 2

;***********************************************************
```

```
;INTRODUCING CORRECTIONS.

;****************************************************************

;COMPUTE TANGENT ELEVATION E0 = G*TF*COS(SF)/(2V)

 LHLD    SF + 2
 XCHG
 LHLD    SF
 COS
 FMUL    D,0D55H
 FMUL    D,NEWT
 FDIV    0D75H,D
 FDIV    V,D
 SHLD    E0
 XCHG
 SHLD    E0 + 2

;****************************************************************

;DRIFT CORRECTION

;**********************************************************MM******

 XCHG
 FMUL    D,0D21H
 SHLD    DRIFT                           ;DRIFT = E0 * DRIFT CONST.
 XCHG
 SHLD    DRIFT +2

;**********************************************************MM**M**

;MUZZLE VELOCITY AND BALLISTIC CORRECTIONS

;**********************************************************MMM**M*

 LHLD    0D2BH                           ;DELV
 XCHG
 LHLD    0D29H
 FMUL    0D31H,D                         ;DELV * C1
 SHLD    TEMP
 XCHG
 SHLD    TEMP + 2
 LHLD    0D2FH                           ;DELC
 XCHG
 LHLD    0D2DH
 FMUL    0D35H,D                         ;DELC * C2
 FADD    TEMP,D
 FADD    D,NEWT
 SHLD    NEWT
 XCHG
 SHLD    NEWT + 2                        ; TF = TF + DELTM
 LHLD    0D2BH                           ;DELV
 XCHG
```

```
         LHLD    0D29H
         FMUL    0D39H,D                     ;DELV * C3
         SHLD    TEMP
         XCHG
         SHLD    TEMP + 2
         LHLD    0D2FH
         XCHG
         LHLD    0D2DH
         FMUL    0D3DH,D                     ;C4*DELC
         FADD    D,TEMP
         FADD    D,SF
         SHLD    SF
         XCHG
         SHLD    SF + 2                      ;SF = SF + DELVM

;*******************************************************MMM**M*:

;WIND CORRECTIONS.

;*******************************************************MMM**M*:

         LHLD    SF + 2
         XCHG
         LHLD    SF
         FADD    D,E0                        ;PHI = SF + E0
         COS
         FMUL    D,WR
         FMUL    D,NEWT                      ;WR *COS(PHI)*TF
         FADD    D,NEWR                      ;RF = RF + DELRW
         LHLD    SF + 2
         XCHG
         LHLD    SF
         COS
         SHLD    COSSF
         XCHG
         SHLD    COSSF + 2
         XCHG
         FMUL    NEWR,D                      ;RF * COS(SF)
         SHLD    TEMP
         XCHG
         SHLD    TEMP + 2
         LHLD    WL + 2
         XCHG
         LHLD    WL
         FMUL    D,NEWT                      ;WL * TF
         FDIV    TEMP,D                      ;WL*TF/(RF*SOS(SF)
         FADD    D,BF                        ;BF = BF + DELAW
         MOV     A,D
         JM      BNEG                        ;BF TO BE 0. TO 360 DEGREES
         FSUB    0D4DH,D                     ;BF - 6,28
         JP      STOB
BNEG:    FADD    0D4DH,D
STOB:    SHLD    BF
         XCHG
         SHLD    BF + 2
```

```
         LHLD     COSSF + 2
         XCHG
         LHLD     COSSF
         FMUL     D,NEWT
         FMUL     D,WR
         FDIV     V,D                           ;DELTW=WR*TF*COS(SF)/V
         FADD     D,NEWT
         SHLD     NEWT
         XCHG
         SHLD     NEWT + 2                      ;TF = TF + DELTW
         LHLD     SF + 2
         XCHG
         LHLD     SF
         FADD     D,E0
         SHLD     TEMP
         XCHG
         SHLD     TEMP + 2
         XCHG
         COS                                    ;COS(PHI)
         SHLD     COSP
         XCHG
         SHLD     COSP + 2
         LHLD     TEMP + 2
         XCHG
         LHLD     TEMP
         SIN                                    ;SIN(PHI)
         FMUL     D,COSP                        ;COS(PHI)*SIN(PHI)
         FMUL     D,V
         SHLD     TEMP
         XCHG
         SHLD     TEMP + 2
         LHLD     COSSF + 2
         XCHG
         LHLD     COSSF
         FMUL     WR,D
         FDIV     TEMP,D
         FSUB     D,SF                          ;SF = SF - DELVW
         SHLD     SF
         XCHG
         SHLD     SF + 2
         XCHG
         FADD     E0,D
         SHLD     GUNEL
         XCHG
         SHLD     GUNEL + 2
         JMP      TIMER                         ;RETURNS TO MAIN ROUTINE

         ;WHEN ITERATIONS TO CONTINUE TO GET SUCCESSIVE VALUES OF RANGE
         ;AND TIME OF FLIGHT WITHIN TOLERANCES

NEXT:    TRANS    NEWR,OLDR                     ;OLDR = NEWR
         DCR      A
         JM       LOST                          ;DOES NOT CONVERGE IN 15
                                                ;ITERATIONS
         TRANS    NEWT,OLDT                     ;OLDT = NEWT
         TRANS    NEWR,OLDR                     ;OLDR = NEWR
```

```
        LHLD    0D03H
        XCHG
        LHLD    0D01H
        FMUL    D,0D11H                 ;RP
        FMUL    D,NEWT                  ;RP * S
        FDIV    NEWR,D
        ARSIN
        SHLD    DV
        XCHG
        SHLD    DV + 2
        XCHG
        FADD    D,0D05H
        SHLD    SF
        XCHG
        SHLD    SF + 2                  ; SF = SP + DV

;LIMIT SF TO BE WITHIN 10 TO 80 DEGREES

        XCHG
        MOV     A,D
        JM      SFNEG
        FSUB    D,0D51H                 ;IF SF .LT. 0
        JM      SFHI                    ; 90 - SF
        FSUB    D,0D51H                 ;SF .GT. 90
        SHLD    SF                      ;RESTORE SF
        XCHG
        SHLD    SF + 2

;SF ADJUSTED BETWEEN 10 TO 80 DEG, WHEN RANGE BELOW 30000 FEET

        LHLD    NEWR + 2
        XCHG
        LHLD    NEWR
        FSUB    D,0D8DH                 ; 30000 - RF
        JM      LOOP
        LHLD    SF + 2
        XCHG
        LHLD    SF
        FSUB    D,0D59H                 ; 10 - SF
        JP      SFTEN
        FADD    D,0D69H                 ; 70 + 10 - SF = 80 - SF
        JM      SFETY
        JMP     LOOP
SFTEN:  TRANS   0D59H,SF                ; SF = 10
        JMP     LOOP
SFETY:  TRANS   0D5DH,SF                ; SF = 80
        JMP     LOOP
SFNEG:  FADD    D,DV                    ; SF = SF + DV - DV
        JMP     LOOP
SFHI:   TRANS   0D61H,SF                ; SF = 89
        JMP     LOOP

;WHEN ROOT RETURNED IS ZERO TRY AGAIN BY ALTERING SF

LOOP2:  LHLD    SF + 2
        XCHG
```

```
        LHLD    SF
        FADD    D,0D65H                 ; SF = SF + 5
        FSUB    D,0D51H                 ;90 - SF
        JM      LOST
        JMP     LOOP
LCST:   XRA     A
        MOV     H,A
        MOV     L,A
        SHLD    NEWR
        XCHG
        SHLD    NEWR + 2
        JMP     TIMER


;***********************************************************************

;ROUTINE TO DETERMINE SQUARE ROOT BY NEWTON - RAPHSON METHOD

;***********************************************************************

;INITIAL GUESS DEPENDS ON WHETHER RP/V IS .GE. V/(G*SIN(SF))
;OR IT IS .LT.V/(G*SIN(SF) ).

;RP/V .GE. V/(G*SIN(SF) ) IMPLIES ROOT IS(-B/(2*A)) + SQROOT

;RP/V .LT. V/(G*SIN(SF)) IMPLIES ROOT IS (-B/(2*A)) - SQROOT

ROOTS:  XRA     A
        STA     FLAG                    ;DECIDES ADD OR SUB SQROOT
        LHLD    BB + 2
        XCHG
        LHLD    BB
        FMUL    D,BB
        SHLD    TEMP
        XCHG
        SHLD    TEMP + 2
        LHLD    AA + 2
        XCHG
        LHLD    AA
        FMUL    D,0DB1H                 ;4 * AA
        FMUL    D,CC                    ;J * AA * CC
        FSUB    D,TEMP                  ;BB*BB - 4*AA*CC
        SHLD    PP
        XCHG
        SHLD    PP + 2
        LHLD    0D03H                   ;CHECK RP/V DIFF V/G*SIN(SF
        XCHG
        LHLD    0D01H
        FDIV    V,D
        SHLD    TEMP
        XCHG
        SHLD    TEMP + 2
        LHLD    AA + 2
        XCHG
```

```
         LHLD    AA
         FDIV    D,V
         FSUB    D,TEMP
         JP      X1R                    ;RP/V - V/G*SIN(SF)
         FSUB    D,0D6DH
                                        ;RP/V .LT. V/(G*SIN(SF))
                                        ;HENCE INITIAL GUESS IS
         JMP     ILOP                   ;V/(G*SIN(SF)) - RP/V
X1R:     MVI     A,1
ILOP:    SHLD    X                      ;FLAG SET,ADD SQROOT
         XCHG                           ;X = INITIAL GUESS
         SHLD    X + 2
         XCHG
         FMUL    D,X
         FSUB    PP,D
         FSUB    D,X                    ;X*X - PP = F(X)
         SHLD    TEMP                   ;X - F(X)
         XCHG
         SHLD    TEMP +2
         LHLD    X + 2
         XCHG
         LHLD    X
         FMUL    D,0D75H
         FDIV    D,TEMP                 ; 2 * X = F'(X)
         SHLD    XX1                    ;X-F(X)/F'(X)
         XCHG
         SHLD    XX1 + 2
         FSUB    X,D
         MOV     A,D                    ;(XX1 - X) = DIFF
         JP      ABSD
         FSUB    D,0D6DH
ABSD:    FSUB    D,0D9DH                ;FOR ABS.VALUE
         JP      OUTL
         TRANS   XX1,X                  ;ITERATIONS COMPLETE
         JMP     ILOP                   ;X = XX1
OUTL:    LDA     FLAG
         JNZ     X1VAL
         LHLD    BB + 2                 ;ADD SQROOT
         XCHG
         LHLD    BB                     ;SUBTRACT SQROOT
         FADD    XX1,D                  ;B + SQROOT
         FDIV    A,D
         FDIV    0D75H,D                ;(B+SQROOT)/(2*A)
         FSUB    D,0D6DH                ;0 -((B+SQROOT)/(2*A))
         SHLD    NEWT
         XCHG
         SHLD    NEWT + 2
         JMP     ROOTR
X1VAL:   LHLD    BB + 2
         XCHG
         LHLD    BB
         FSUB    D,0D6DH                ;0 - B = -B
         FADD    D,XX1
         FDIV    A,D
         FDIV    0D75H,D
```

```
        XCHG
        SHLD    NEWT + 2
        JMP     ROOTR


       ;END OF PROGRAM
N:      DS      1
JSTAR:  DS      1
P:      DS      1
K:      DS      1
L:      DS      1
M:      DS      1
M1:     DS      1
M2:     DS      1
NN:     DS      1
FLAG:   DS      1
SP1:    DS      2
SP2:    DS      2
SP3:    DS      2
SP4:    DS      2
IPT1:   DS      2
IPT2:   DS      2
IPT3:   DS      2
IPT4:   DS      2
STAC1:  DS      2
STAC2:  DS      2
STAC3:  DS      2
STAC4:  DS      2
CLOCK:  DS      4
WR:     DS      4
WL:     DS      4
V:      DS      4
OLDR:   DS      4
SF:     DS      4
OLDT:   DS      4
AA:     DS      4
BB:     DS      4
CC:     DS      4
TEMP:   DS      4
PP:     DS      4
X:      DS      4
XX1:    DS      4
NEWT:   DS      4
NEWR:   DS      4
BF:     DS      4
EO:     DS      4
DRIFT:  DS      4
COSSF:  DS      4
COSP:   DS      4
GUNEL:  DS      4
DV:     DS      4
DRX:    DS      4
DSX:    DS      4
DBX:    DS      4
CLCK1:  DS      4
ADR1::  DS      4
```

```
A1R1:     S        4
A0E1:    DS        4
A0B1:    DS        4
A1B1:    DS        4
A0R2:    DS        4
A1R2:    DS        4
A0E2:    DS        4
A1E2:    DS        4
A0B2     DS        4
A1B2     DS        4
A0R3:    DS        4
A1R3:    DS        4
A0E3:    DS        4
A1E3:    DS        4
A0B3:    DS        4
A1B3:    DS        4
AAR:     DS        4
AAE:     DS        4
AAB:     DS        4
BBR:     DS        4
BBE:     DS        4
BBB:     DS        4
```